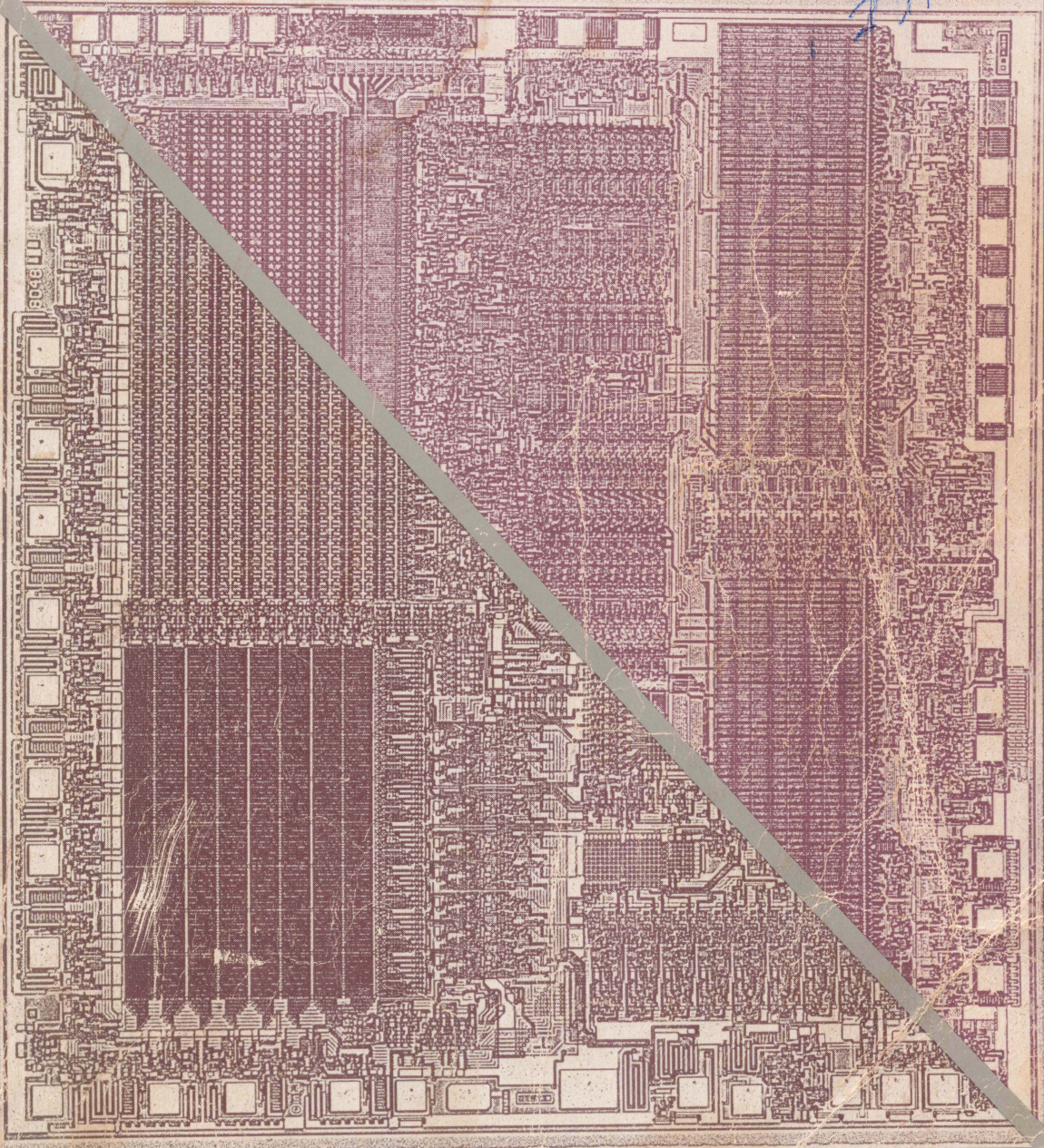


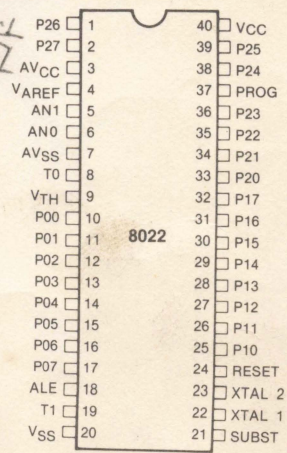
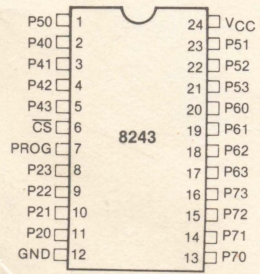
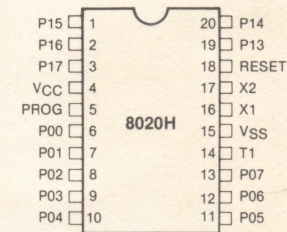
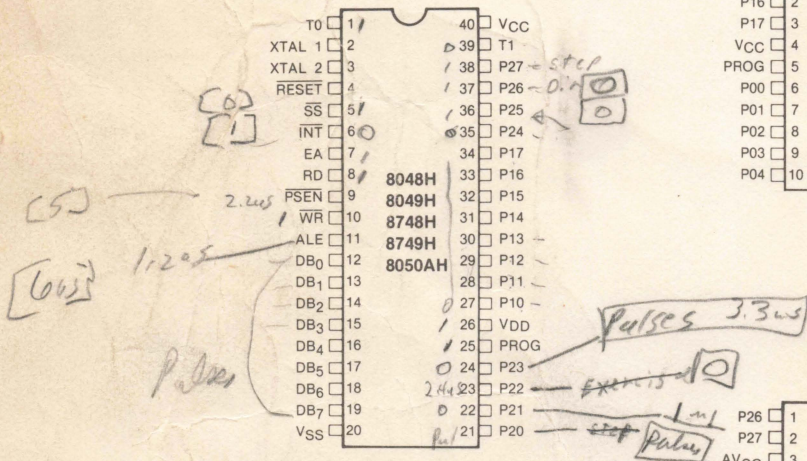
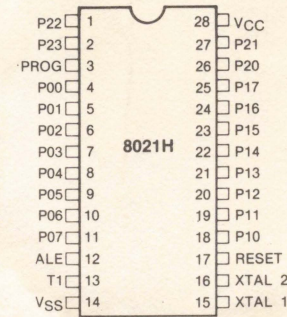
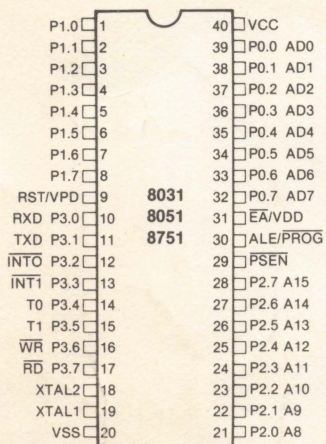
Intel®

# Microcontroller User's Manual

Ken - 701









21



23 22 24 )

23

# MICROCONTROLLER USER'S MANUAL

27 Pulse

37ms

Pin 38

3.8ms

Pin 29

MAY 1982

000001

36

27 H

26 H

P25 L

28 H

23

22

21

20

40



INTEL  
MICROCONTROLLER  
USER'S MANUAL

MAY 1982

Intel Corporation makes no warranty for the use of its products and assumes no responsibility for any errors which may appear in this document nor does it make a commitment to update the information contained herein.

The following are trademarks of Intel Corporation and may only be used to identify Intel Products:

BXP, CREDIT, i, ICE, iCS, i<sub>m</sub>, iMMX, Insite, Intel, int<sub>el</sub>, Inteleview,  
intellec, iOSP, iRMX, iSBC, iSBX, Library Manager, MCS,  
Megachassis, Micromainframe, Micromap, Multimodule,  
Plug-A-Bubble, PROMPT, RMX/80, System 2000 and UPI.

MDS is an ordering code only and is not used as a product name or trademark. MDS® is a registered trademark of Mohawk Data Sciences Corporation.

\*MULTIBUS is a patented Intel bus.

Additional copies of this manual or other Intel literature may be obtained from:

Intel Corporation  
Literature Department SV3-3  
3065 Bowers Avenue  
Santa Clara, CA 95051



# Table of Contents

## CHAPTER 1

### Introduction

|     |   |     |
|-----|---|-----|
| 1.0 | Introduction to the MCS®-48 Family                            | 1-1 |
| 1.1 | Introduction to the MCS®-51 Family                            | 1-2 |
| 1.2 | Intel Microcontroller Development System and Software Support | 1-3 |

## CHAPTER 2

### The Single Component MCS®-48 System

|     |  |      |
|-----|--|------|
| 2.0 | Introduction                             | 2-1  |
| 2.1 | Architecture                             | 2-1  |
| 2.2 | Pin Description                          | 2-18 |
| 2.3 | Programming, Verifying and Erasing EPROM | 2-18 |
| 2.4 | Reading Internal Program Memory          | 2-23 |
| 2.5 | 8020H/8021H Functional Specifications    | 2-23 |
| 2.6 | 8022 Functional Specifications           | 2-31 |

## CHAPTER 3

### Expanded MCS®-48 System

|     |                             |      |
|-----|-----------------------------|------|
| 3.0 | Introduction                | 3-1  |
| 3.1 | Expansion of Program Memory | 3-1  |
| 3.2 | Expansion of Data Memory    | 3-3  |
| 3.3 | Expansion of Input/Output   | 3-5  |
| 3.4 | Multi-Chip MCS-48 Systems   | 3-10 |
| 3.5 | Memory Bank Switching       | 3-11 |
| 3.6 | Control Signal Summary      | 3-11 |
| 3.7 | Port Characteristics        | 3-11 |

## CHAPTER 4

### MCS®-48 Instruction Set

|     |                             |     |
|-----|-----------------------------|-----|
| 4.0 | Introduction                | 4-1 |
| 4.1 | Instruction Set Description | 4-5 |

## CHAPTER 5

### MCS®-48 Application Examples

|     |  |      |
|-----|--|------|
| 5.0 | Introduction                               | 5-1  |
| 5.1 | Hardware Examples                          | 5-1  |
| 5.2 | I/O Expansion Techniques                   | 5-11 |
| 5.3 | 8049H Emulator Circuit Description - 6 MHz | 5-18 |
| 5.4 | Software Examples                          | 5-23 |

## CHAPTER 6

### MCS®-51 Architecture

|     |                              |     |
|-----|------------------------------|-----|
| 6.0 | Memory Organization          | 6-2 |
| 6.1 | Special Function Registers   | 6-2 |
| 6.2 | Oscillator and Clock Circuit | 6-4 |



|      |                                 |      |
|------|---------------------------------|------|
| 6.3  | CPU Timing .....                | 6-6  |
| 6.4  | Port Operation .....            | 6-6  |
| 6.5  | Accessing External Memory ..... | 6-9  |
| 6.6  | Timers .....                    | 6-12 |
| 6.7  | Serial Interface .....          | 6-16 |
| 6.8  | Interrupts .....                | 6-26 |
| 6.9  | RST/VPD Pin .....               | 6-30 |
| 6.10 | 8751 .....                      | 6-31 |
| 6.11 | Family Pin Description .....    | 6-33 |

## CHAPTER 7

### MCS®-51 Memory Organization, Addressing Modes, and Data Manipulation

|     |                                |      |
|-----|--------------------------------|------|
| 7.0 | Memory Organization .....      | 7-1  |
| 7.1 | Operand Addressing .....       | 7-3  |
| 7.2 | Data Manipulation .....        | 7-5  |
| 7.3 | Boolean Processor .....        | 7-6  |
| 7.4 | Data Transfer Operations ..... | 7-6  |
| 7.5 | Logic Operations .....         | 7-8  |
| 7.6 | Arithmetic Operations .....    | 7-9  |
| 7.7 | Control Transfer .....         | 7-11 |

## CHAPTER 8

### MCS®-51 Instruction Set

|     |   |      |
|-----|---|------|
| 8.0 | What the Instruction Set Is .....         | 8-1  |
| 8.1 | Organization of the Instruction Set ..... | 8-1  |
| 8.2 | Instruction Definitions .....             | 8-10 |

## CHAPTER 9

### MCS®-51 Application Examples

|     |   |      |
|-----|---|------|
| 9.0 | 8051 Programming Techniques .....       | 9-1  |
| 9.1 | Peripheral Interfacing Techniques ..... | 9-13 |
| 9.2 | Connecting to Peripherals .....         | 9-22 |

## CHAPTER 10

### Microcontroller Specifications

|   |       |
|---|-------|
| 8020H HMOS Single Component 8-Bit Microcomputer .....                         | 10-1  |
| 8021H HMOS Single Component 8-Bit Microcomputer .....                         | 10-4  |
| 8022 Single Component 8-Bit Microcomputer<br>with On-Chip A/D Converter ..... | 10-8  |
| 8048H/8748H/8035HL/8049H/8749H/8039HL/8050AH/8040AHL .....                    | 10-14 |
| 8748/8035 Single Component 8-Bit Microcomputer .....                          | 10-24 |
| 8243 MCS®-48 Input/Output Expander - EXPRESS .....                            | 10-33 |
| Single Component 8-Bit Microcomputers - EXPRESS .....                         | 10-39 |
| M8048/M8748/M8035HL Single Component 8-Bit Microcomputer - MILITARY .....     | 10-43 |
| Dash Numbers and Their Meaning .....  | 10-51 |
| 80C48/80C35/80C49/80C39 CHMOS Single Component 8-Bit Microcomputers .....     | 10-52 |

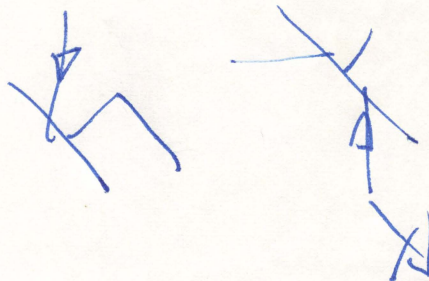


---

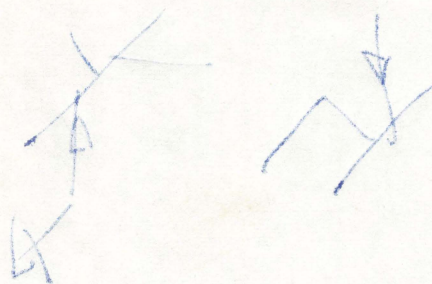
# Introduction

1

---









# CHAPTER 1 INTRODUCTION

## 1.0 INTRODUCTION TO THE MCS®-48 FAMILY

In 1980 Intel increased the performance of the MCS-48 family by converting much of the product line to HMOS technology. HMOS is a high-performance N-MOS technology developed by Intel. Some of the technical improvements are highlighted in Table 1-2.

Since 1976 Intel has offered products for the full range of single-chip microcomputer applications by pushing the 8048's architecture and technology in several directions. The 8749/8049 runs nearly twice as fast as the 8748/8048 while doubling the amount of on-

chip program memory and data memory. The recently introduced 8050AH again doubles the on-chip RAM and ROM of the 8049. Applications requiring solely external program memory are satisfied with the 8035 and 8039. Cost-sensitive and less I/O intensive applications incorporate the 8021 which executes a subset of the 8048's instruction set at a slower speed. The 8022 integrated an 8-bit A/D converter onto the 8021 die to allow the chip to interface directly to a world in which most signals are analog. Finally, the 8020H put the 8021H CPU performance into a 20-pin, 300 mil wide package for space-critical applications. Table 1-1 positions these products relative to their on-chip features.

**Table 1-1. MCS-48 Product On-Chip Features**

|                      | 8050H          | 8749H           | 8049H           | 8748            | 8048H         | 8022          | 8021H         | 8020H         |
|----------------------|----------------|-----------------|-----------------|-----------------|---------------|---------------|---------------|---------------|
| 8-Bit CPU            | ✓              | ✓               | ✓               | ✓               | ✓             | ✓             | ✓             | ✓             |
| Program Memory       | 4K x 8<br>ROM  | 2K x 8<br>EPROM | 2K x 8<br>EPROM | 1K x 8<br>EPROM | 1K x 8<br>ROM | 2K x 8<br>ROM | 1K x 8<br>ROM | 1K x 8<br>ROM |
| Data Memory          | 256 x 8<br>RAM | 128 x 8<br>RAM  | 128 x 8<br>RAM  | 64 x 8<br>RAM   | 64 x 8<br>RAM | 64 x 8<br>RAM | 64 x 8<br>RAM | 64 x 8<br>RAM |
| I/O Lines            | 27             | 27              | 27              | 27              | 27            | 28            | 21            | 13            |
| 8-Bit Timer/Counter  | ✓              | ✓               | ✓               | ✓               | ✓             | ✓             | ✓             | ✓             |
| Reset                | ✓              | ✓               | ✓               | ✓               | ✓             | ✓             | ✓             | ✓             |
| Interrupt            | ✓              | ✓               | ✓               | ✓               | ✓             | ✓             | ✓             | ✓             |
| Oscillator and Clock | ✓              | ✓               | ✓               | ✓               | ✓             | ✓             | ✓             | ✓             |

In 1980 Intel increased the performance of the MCS-48 family by converting much of the product line to HMOS technology. HMOS is a high-performance N-MOS technology developed by Intel. Some of the technical improvements are highlighted in Table 1-2.

**Table 1-2. Comparison Between 8048H  
and 8048**

|                                    | Industry's<br>8048 | Intel's<br>8048H |
|------------------------------------|--------------------|------------------|
| Cycle Time ( $\mu$ sec)            | 2.5                | 1.9              |
| ICC (typ, mA)                      | 65                 | 40               |
| VDD (Standby<br>voltage) min V     | 5                  | 2.2              |
| IDD (Standby cur-<br>rent, max) mA | 15                 | 8                |



# INTRODUCTION

This transition of processes, the popularity of the architecture and the learning curve phenomena continue to make the MCS-48 family a cost-effective solution for tomorrow's designs.

The popularity of this line of single chips is also due to the support tools Intel has provided to the marketplace. EPROM microcomputers continue to be indispensable tools in prototyping and limited production. Training courses bring skilled instructors to teach the basics of the MCS-48 family to help you design your end product quickly. In-circuit emulation provides aid between the difficult bridge of software and hardware debugging. The literature, consisting of application notes, software programs, data sheets and user manuals, is a powerful tool in learning the unique features of the MCS-48 architecture.

## 1.1 INTRODUCTION TO MCS®-51

The goal of the MCS-51 family is to extend the architecture of the industry standard MCS-48 family into the 80s. This meant increasing the performance of the MCS-48's CPU as well as increasing the power, variety and quantity of on-chip CPU peripherals.

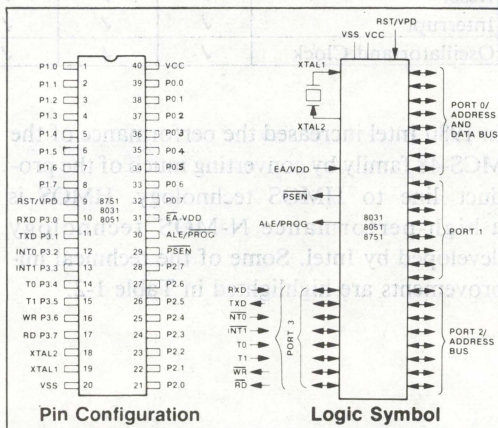
As a result, the MCS-51 family addresses applications that require a high degree of on-chip functionality. It is the highest performance microcomputer family in the world and outperforms all microprocessors and microcomputers in control-oriented applications. Its enhanced architecture offers an upward compatible growth path for MCS-48 users. Table 1-3 outlines these features.

**Table 1-3. 8051 Functions/Speed Relative to 8048**

- 4X Program Memory (4k Bytes)
- 2X Data Memory (128 Bytes)
- 2X Register Banks (4 vs. 2)
- 2X Timers (Two 16-bit Timers)
- New Full-Duplex Serial I/O Port
- More I/O Pins (32 vs. 27)
- Enhanced MCS-48 Architecture
- 2½X to 10X Execution Speed

The MCS-51 family currently consists of three products: the 8051, the 8031, and the 8751.

The 8051 is a stand-alone high-performance single-chip computer intended for use in sophisticated real-time applications such as instrumentation, industrial control and intelligent computer peripherals. It provides the hardware features, architectural enhancements and new instructions that make it a powerful and cost effective controller for applications requiring up to 64K-bytes of program memory and/or up to 64K-bytes of data storage. A Logic Symbol is shown in Figure 1-1.



**Figure 1-1**



## INTRODUCTION

---

The 8031 is a control-oriented CPU without on-chip program memory. It can address 64K-bytes of External Program Memory in addition to 64K-bytes of External Data Memory. For systems requiring extra capability, each member of the MCS-51 family can be expanded using standard memories and the byte oriented MCS-80 and MCS-85 peripherals. The 8051 is an 8031 with the lower 4K-bytes of Program Memory filled with the on-chip mask programmable ROM while the 8751 has 4K-bytes of UV-light-erasable/electrically-programmable ROM.

These three pin-compatible versions of the MCS-51 family (8051,8031,8751) reduce development problems to a minimum and provide maximum flexibility. The 8751 is well suited for development, prototyping, low-volume production and applications requiring field updates; the 8051 for low-cost, high-volume production; and the 8031 for applications desiring the flexibility of external Program Memory which can easily be modified and updated in the field.

### 1.2 INTEL MICROCONTROLLER DEVELOPMENT SYSTEM AND SOFTWARE SUPPORT

- ASM-48 Absolute macro assembler for the 8048H/8049H/8050H/8022/8021H/8020H
- ICE-49 Real-time in-circuit emulator for 8048H/8049H
- ICE-22 Real-time in-circuit emulator for 8022/8021H/8020H
- EM-2 8022/8021H/8020H emulator board
- HSE-49 Single board 11 MHz evaluation tool with limited real-time emulation
- UPP-848 8748 personality card for UPP-103 Universal PROM Programmer
- UPP-549 8749/8748H PROM programmer adaptor socket
- ASM51 Absolute macro assembler for the 8051.
- CONV51 8048 assembly language source code to 8051 assembly source code conversion program.
- EM-51 8051/8751 emulator board that uses a modified 8051 and an EPROM.
- ICE-51™ Real-time in-circuit emulator.
- SDK-51 System design kit for developing user Prototype around the 8051.
- UPP-551 8751 personality card for UPP-103 Universal PROM Programmer.
- 8051 Workshop.







---

# **The Single Component MCS<sup>®</sup>-48 System**

---

**2**



---

# MCS®-48 System The Single Component

---

2



# CHAPTER 2

## THE SINGLE COMPONENT MCS®-48 SYSTEM

### 2.0 INTRODUCTION

Sections 2.1 through 2.4 describe in detail the functional characteristics of the 8748 and 8749H EPROM, 8748H/8049H/8050H ROM, and 8035/8035HL/8039HL/8040HL CPU only single component microcomputers. Unless otherwise noted, details within these sections apply to all versions. Section 2.5 describes the operation of the 8020H/8021H and Section 2.6 describes the 8022. This chapter is limited to those functions useful in single-chip implementations of the MCS-48. Chapter 3 discusses functions which allow expansion of program memory, data memory, and input output capability.

### 2.1 ARCHITECTURE

The following sections break the MCS-48 Family into functional blocks and describe each in detail. The following description will use the 8048H as the representative product for the family. See Figure 2-1.

#### 2.1.1 Arithmetic Section

The arithmetic section of the processor contains the basic data manipulation functions of the 8048H and can be divided into the following blocks:

- Arithmetic Logic Unit (ALU)
- Accumulator
- Carry Flag
- Instruction Decoder

In a typical operation data stored in the accumulator is combined in the ALU with data from another source on the internal bus (such as a register or I/O port) and the result is stored in the accumulator or another register.

The following is more detailed description of the function of each block.

#### INSTRUCTION DECODER

The operation code (op code) portion of each program instruction is stored in the Instruction Decoder and converted to outputs which control the function of each of the blocks of the Arithmetic Section. These lines control the source of data and the destination register as well as the function performed in the ALU.

#### ARITHMETIC LOGIC UNIT

The ALU accepts 8-bit data words from one or two sources and generates an 8-bit result under control of the Instruction Decoder. The ALU can perform the following functions:

- Add With or Without Carry
- AND, OR, Exclusive OR
- Increment/Decrement
- Bit Complement
- Rotate Left, Right
- Swap Nibbles
- BCD Decimal Adjust

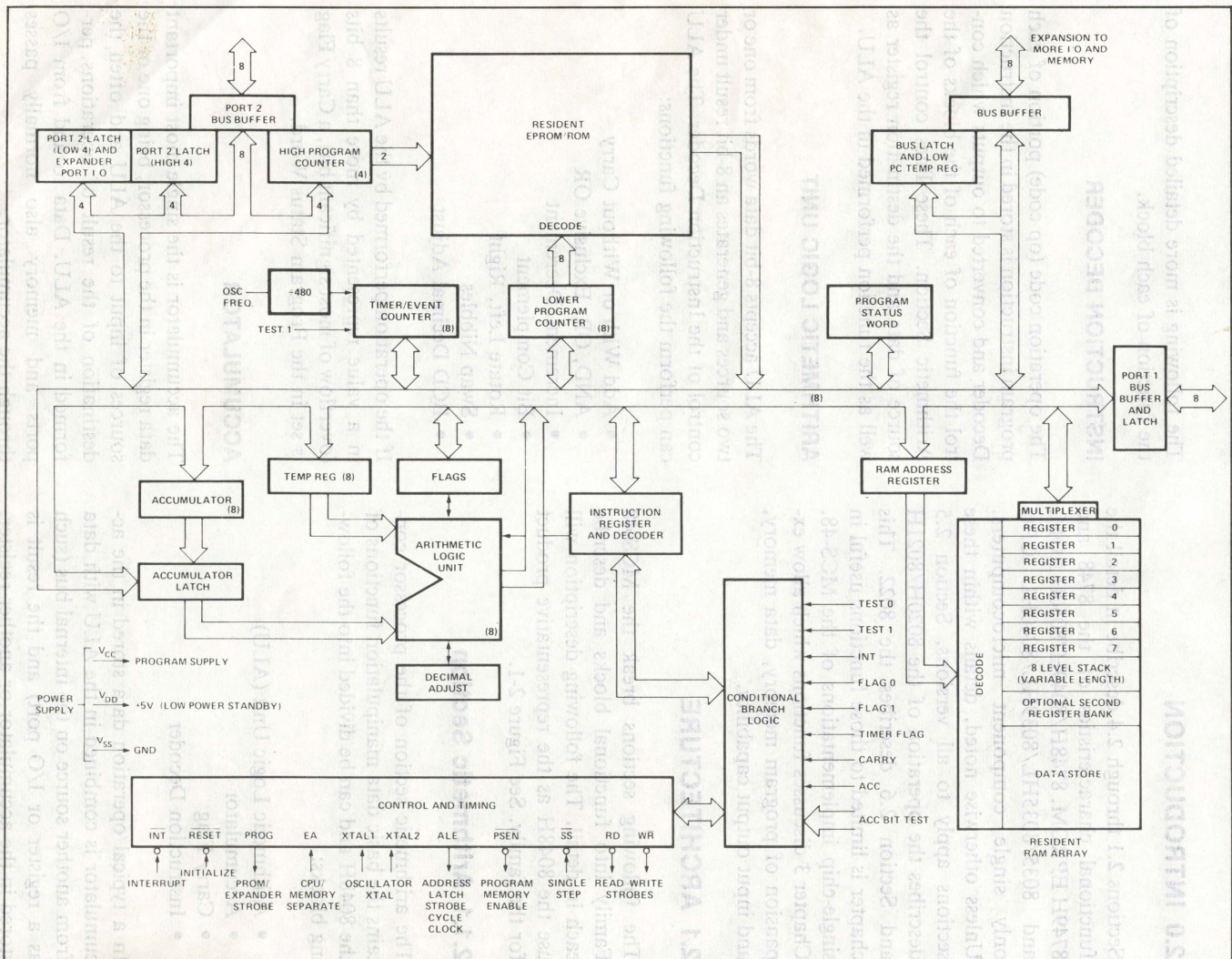
If the operation performed by the ALU results in a value represented by more than 8 bits (overflow of most significant bit), a Carry Flag is set in the Program Status Word.

#### ACCUMULATOR

The accumulator is the single most important data register in the processor, being one of the sources of input to the ALU and often the destination of the result of operations performed in the ALU. Data to and from I/O ports and memory also normally passes through the accumulator.



## SINGLE COMPONENT MCS-48 SYSTEM





## SINGLE COMPONENT MCS-48 SYSTEM

### 2.1.2 Program Memory

Resident program memory consists of 1024, 2048, or 4096 words eight bits wide which are addressed by the program counter. In the 8748 and the 8749H this memory is user programmable and erasable EPROM; in the 8048H/8049H/8050H the memory is ROM which is mask programmable at the factory. The 8035/8035HL/8039HL/8040HL has no internal program memory and is used with external memory devices. Program code is completely interchangeable among the various versions. To access the upper 2K of program memory in the 8050H, and other MCS-48 devices, a select memory bank and a JUMP or CALL instruction must be executed to cross the 2K boundary. See Section 2.3 for EPROM programming techniques.

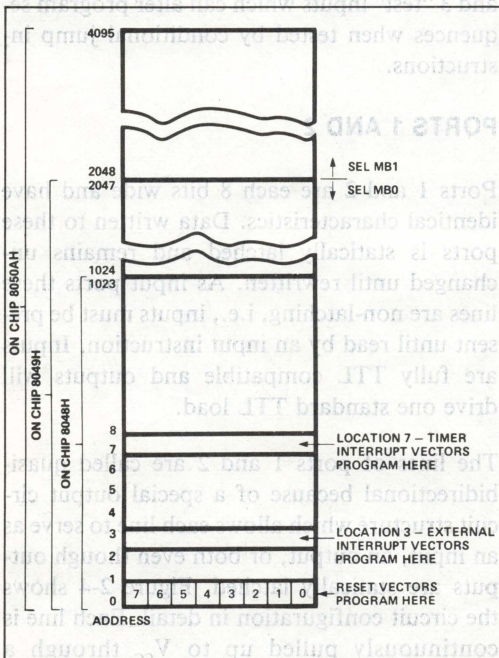


Figure 2-2. Program Memory Map

There are three locations in Program Memory of special importance as shown in Figure 2-2.

#### LOCATION 0

Activating the Reset line of the processor causes the first instruction to be fetched from location 0.

#### LOCATION 3

Activating the Interrupt input line of the processor (if interrupt is enabled) causes a jump to subroutine at location 3.

#### LOCATION 7

A timer/counter interrupt resulting from timer/counter overflow (if enabled) causes a jump to subroutine at location 7.

Therefore, the first instruction to be executed after initialization is stored in location 0, the first word of an external interrupt service

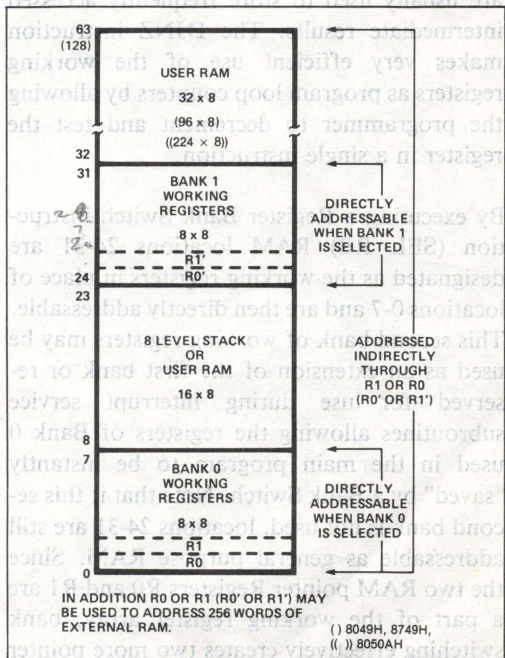


Figure 2-3. Data Memory Map



## SINGLE COMPONENT MCS-48 SYSTEM

subroutine is stored in location 2, and the first word of a timer/counter service routines is stored in location 7. Program memory can be used to store constants as well as program instructions. Instructions such as MOVP and MOVP3 allow easy access to data "lookup" tables.

### 2.1.3 Data Memory

Resident data memory is organized as 64, 128, or 256 words 8-bits wide in the 8048H, 8049H and 8050H. All locations are indirectly addressable through either of two RAM Pointer Registers which reside at address 0 and 1 of the register array. In addition, as shown in Figure 2-3, the first 8 locations (0-7) of the array are designated as working registers and are directly addressable by several instructions. Since these registers are more easily addressed, they are usually used to store frequently accessed intermediate results. The DJNZ instruction makes very efficient use of the working registers as program loop counters by allowing the programmer to decrement and test the register in a single instruction.

By executing a Register Bank Switch instruction (SEL RB) RAM locations 24-31 are designated as the working registers in place of locations 0-7 and are then directly addressable. This second bank of working registers may be used as an extension of the first bank or reserved for use during interrupt service subroutines allowing the registers of Bank 0 used in the main program to be instantly "saved" by a Bank Switch. Note that if this second bank is not used, locations 24-31 are still addressable as general purpose RAM. Since the two RAM pointer Registers R0 and R1 are a part of the working register array, bank switching effectively creates two more pointer registers (R0' and R1') which can be used with

R0 and R1 to easily access up to four separate working areas in RAM at one time. RAM locations (8-23) also serve a dual role in that they contain the program counter stack as explained in Section 2.1.6. These locations are addressed by the Stack Pointer during subroutine calls as well as by RAM Pointer Registers R0 and R1. If the level of subroutine nesting is less than 8, all stack registers are not required and can be used as general purpose RAM locations. Each level of subroutine nesting not used provides the user with two additional RAM locations.

### 2.1.4 Input/Output

The 8048H has 27 lines which can be used for input or output functions. These lines are grouped as 3 ports of 8 lines each which serve as either inputs, outputs or bidirectional ports and 3 "test" inputs which can alter program sequences when tested by conditional jump instructions.

#### PORTS 1 AND 2

Ports 1 and 2 are each 8 bits wide and have identical characteristics. Data written to these ports is statically latched and remains unchanged until rewritten. As input ports these lines are non-latching, i.e., inputs must be present until read by an input instruction. Inputs are fully TTL compatible and outputs will drive one standard TTL load.

The lines of ports 1 and 2 are called quasi-bidirectional because of a special output circuit structure which allows each line to serve as an input, an output, or both even though outputs are statically latched. Figure 2-4 shows the circuit configuration in detail. Each line is continuously pulled up to  $V_{CC}$  through a resistive device of relatively high impedance.



## SINGLE COMPONENT MCS-48 SYSTEM

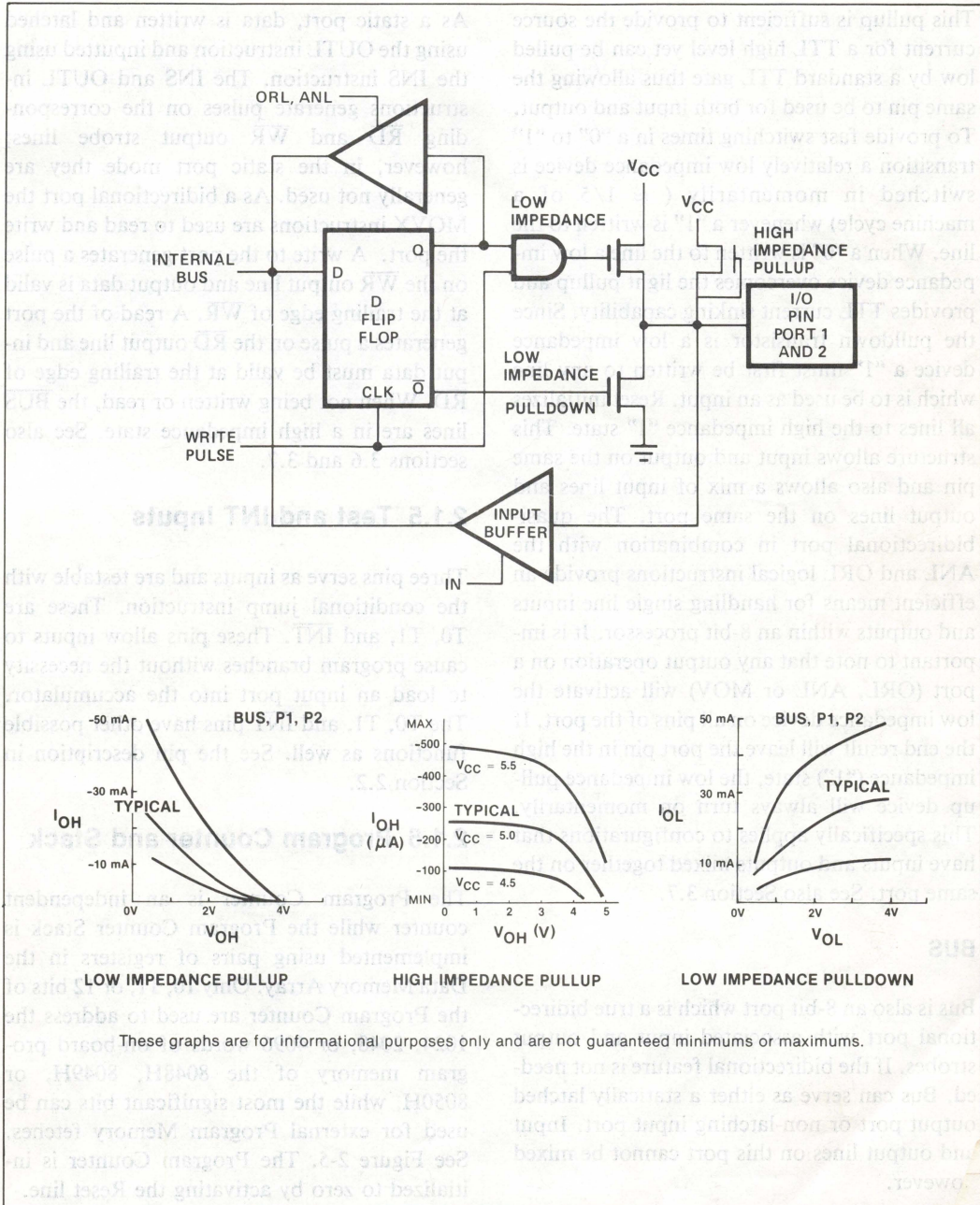


Figure 2-4. "Quasi-bidirectional" Port Structure



This pullup is sufficient to provide the source current for a TTL high level yet can be pulled low by a standard TTL gate thus allowing the same pin to be used for both input and output. To provide fast switching times in a "0" to "1" transition a relatively low impedance device is switched in momentarily ( $\approx 1/5$  of a machine cycle) whenever a "1" is written to the line. When a "0" is written to the line a low impedance device overcomes the light pullup and provides TTL current sinking capability. Since the pulldown transistor is a low impedance device a "1" must first be written to any line which is to be used as an input. Reset initializes all lines to the high impedance "1" state. This structure allows input and output on the same pin and also allows a mix of input lines and output lines on the same port. The quasi-bidirectional port in combination with the ANL and ORL logical instructions provide an efficient means for handling single line inputs and outputs within an 8-bit processor. It is important to note that any output operation on a port (ORL, ANL or MOV) will activate the low impedance device on all pins of the port. If the end result will leave the port pin in the high impedance ("1") state, the low impedance pull-up device will always turn on momentarily. This specifically applies to configurations that have inputs and outputs mixed together on the same port. See also Section 3.7.

### BUS

Bus is also an 8-bit port which is a true bidirectional port with associated input and output strobes. If the bidirectional feature is not needed, Bus can serve as either a statically latched output port or non-latching input port. Input and output lines on this port cannot be mixed however.

As a static port, data is written and latched using the OUTL instruction and inputted using the INS instruction. The INS and OUTL instructions generate pulses on the corresponding  $\overline{RD}$  and  $\overline{WR}$  output strobe lines; however, in the static port mode they are generally not used. As a bidirectional port the MOVX instructions are used to read and write the port. A write to the port generates a pulse on the  $\overline{WR}$  output line and output data is valid at the trailing edge of  $\overline{WR}$ . A read of the port generates a pulse on the  $\overline{RD}$  output line and input data must be valid at the trailing edge of  $\overline{RD}$ . When not being written or read, the  $\overline{BUS}$  lines are in a high impedance state. See also sections 3.6 and 3.7.

### 2.1.5 Test and INT Inputs

Three pins serve as inputs and are testable with the conditional jump instruction. These are T0, T1, and  $\overline{INT}$ . These pins allow inputs to cause program branches without the necessity to load an input port into the accumulator. The T0, T1, and  $\overline{INT}$  pins have other possible functions as well. See the pin description in Section 2.2.

### 2.1.6 Program Counter and Stack

The Program Counter is an independent counter while the Program Counter Stack is implemented using pairs of registers in the Data Memory Array. Only 10, 11, or 12 bits of the Program Counter are used to address the 1024, 2048, or 4096 words of on-board program memory of the 8048H, 8049H, or 8050H, while the most significant bits can be used for external Program Memory fetches. See Figure 2-5. The Program Counter is initialized to zero by activating the Reset line.



```

MOV A, PSW
ANL A, #7Fh
ADD 8

```

## SINGLE COMPONENT MCS-48 SYSTEM

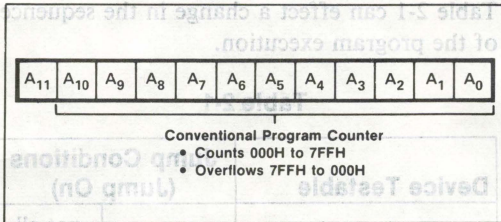


Figure 2-5. Program Counter

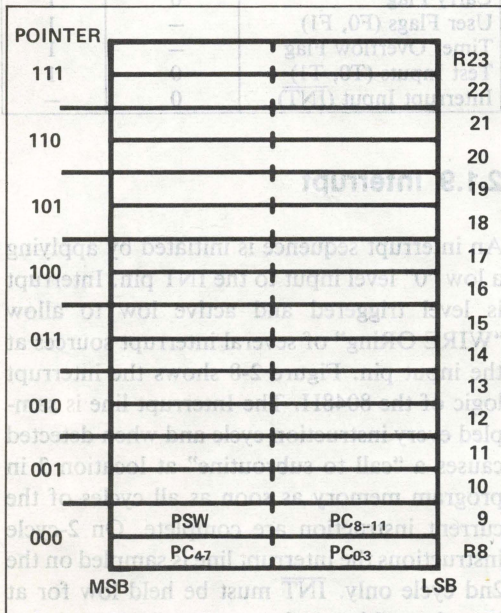


Figure 2-6. Program Counter Stack

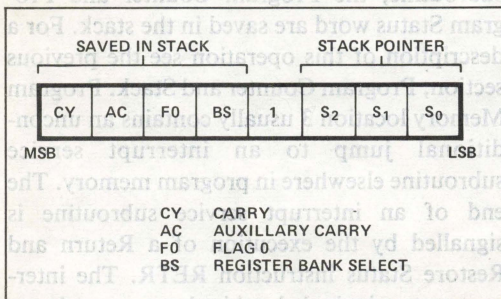


Figure 2-7. Program Status Word (PSW)

An interrupt or CALL to a subroutine causes the contents of the program counter to be stored in one of the 8 register pairs of the Program Counter Stack as shown in Figure 2-6. The pair to be used is determined by a 3-bit Stack Pointer which is part of the Program Status Word (PSW). Data RAM locations 8-23 are available as stack registers and are used to store the Program Counter and 4 bits of PSW as shown in the figure. The Stack Pointer when initialized to 000 points to RAM locations 8 and 9. The first subroutine jump or interrupt results in the program counter contents being transferred to locations 8 and 9 of the RAM array. The stack pointer is then incremented by one to point to locations 10 and 11 in anticipation of another CALL. Nesting of subroutines within subroutines can continue up to 8 times without overflowing the stack. If overflow does occur the deepest address stored (locations 8 and 9) will be overwritten and lost since the stack pointer overflows from 111 to 000. It also underflows from 000 to 111.

The end of a subroutine, which is signalled by a return instruction (RET or RETR), causes the Stack Pointer to be decremented and the contents of the resulting register pair to be transferred to the Program Counter.

### 2.1.7 Program Status Word

An 8-bit status word which can be loaded to and from the accumulator exists called the Program Status Word (PSW). The accompanying figure shows the information available in the word. The Program Status Word is actually a collection of flip-flops throughout the machine which can be read or written as a whole. The ability to write to PSW allows for easy restoration of machine state after a power down sequence.



## SINGLE COMPONENT MCS-48 SYSTEM

The upper four bits of PSW are stored in the Program Counter Stack with every call to subroutine or interrupt vector and are optionally restored upon return with the RETR instruction. The RET return instruction does not update PSW.

The PSW bit definitions are as follows:

Bits 0-2: Stack Pointer bits ( $S_0$ ,  $S_1$ ,  $S_2$ )

Bit 3: Not used ("1" level when read)

Bit 4: Working Register Bank Switch Bit (BS)

0 = Bank 0

1 = Bank 1

Bit 5: Flag 0 bit ( $F_0$ ) user controlled flag which can be complemented or cleared, and tested with the conditional jump instruction JF0.

Bit 6: Auxiliary Carry (AC) carry bit generated by an ADD instruction and used by the decimal adjust instruction DA A.

Bit 7: Carry (CY) carry flag which indicates that the previous operation has resulted in overflow of the accumulator.

### 2.1.8 Conditional Branch Logic

The conditional branch logic within the processor enables several conditions internal and external to the processor to be tested by the users program. By using the conditional jump instruction the conditions that are listed in

Table 2-1 can effect a change in the sequence of the program execution.

Table 2-1

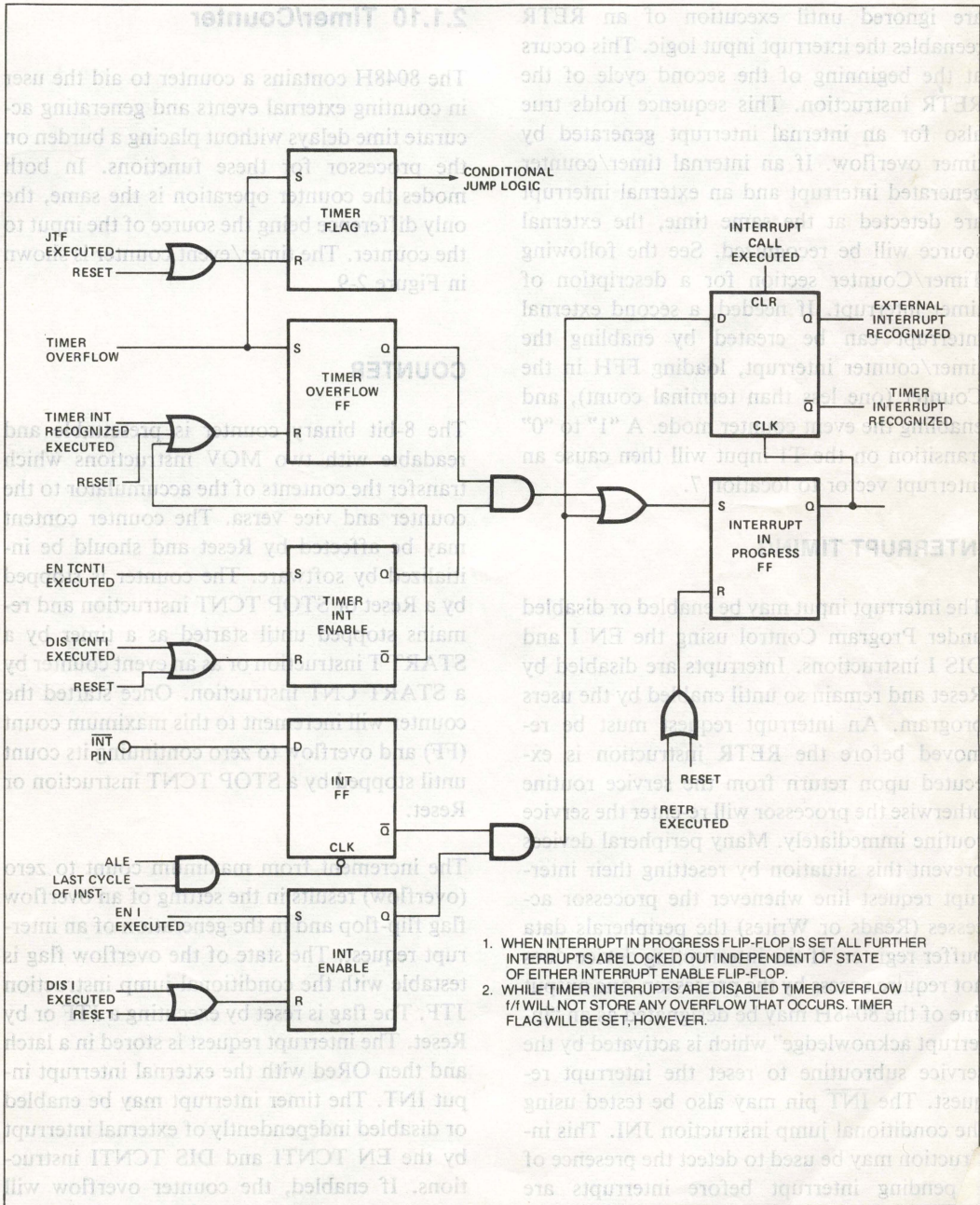
| Device Testable                      | Jump Conditions (Jump On) |               |
|--------------------------------------|---------------------------|---------------|
|                                      | All zeros                 | not all zeros |
| Accumulator                          | —                         | 1             |
| Accumulator Bit                      | —                         | 1             |
| Carry Flag                           | 0                         | 1             |
| User Flags ( $F_0$ , $F_1$ )         | —                         | 1             |
| Timer Overflow Flag                  | —                         | 1             |
| Test Inputs ( $T_0$ , $T_1$ )        | 0                         | 1             |
| Interrupt Input ( $\overline{INT}$ ) | 0                         | —             |

### 2.1.9 Interrupt

An interrupt sequence is initiated by applying a low "0" level input to the INT pin. Interrupt is level triggered and active low to allow "WIRE ORing" of several interrupt sources at the input pin. Figure 2-8 shows the interrupt logic of the 8048H. The Interrupt line is sampled every instruction cycle and when detected causes a "call to subroutine" at location 3 in program memory as soon as all cycles of the current instruction are complete. On 2-cycle instructions the interrupt line is sampled on the 2nd cycle only.  $\overline{INT}$  must be held low for at least 3 machine cycles to ensure proper interrupt operations. As in any CALL to subroutine, the Program Counter and Program Status word are saved in the stack. For a description of this operation see the previous section, Program Counter and Stack. Program Memory location 3 usually contains an unconditional jump to an interrupt service subroutine elsewhere in program memory. The end of an interrupt service subroutine is signalled by the execution of a Return and Restore Status instruction RETR. The interrupt system is single level in that once an interrupt is detected all further interrupt requests



# 2.5 3 75 7.5x10<sup>6</sup> SINGLE COMPONENT MCS-48 SYSTEMS



**Figure 2-8. Interrupt Logic**



are ignored until execution of an RETR reenables the interrupt input logic. This occurs at the beginning of the second cycle of the RETR instruction. This sequence holds true also for an internal interrupt generated by timer overflow. If an internal timer/counter generated interrupt and an external interrupt are detected at the same time, the external source will be recognized. See the following Timer/Counter section for a description of timer interrupt. If needed, a second external interrupt can be created by enabling the timer/counter interrupt, loading FFH in the Counter (one less than terminal count), and enabling the event counter mode. A "1" to "0" transition on the T1 input will then cause an interrupt vector to location 7.

## INTERRUPT TIMING

The interrupt input may be enabled or disabled under Program Control using the EN I and DIS I instructions. Interrupts are disabled by Reset and remain so until enabled by the users program. An interrupt request must be removed before the RETR instruction is executed upon return from the service routine otherwise the processor will re-enter the service routine immediately. Many peripheral devices prevent this situation by resetting their interrupt request line whenever the processor accesses (Reads or Writes) the peripherals data buffer register. If the interrupting device does not require access by the processor, one output line of the 8048H may be designated as an "interrupt acknowledge" which is activated by the service subroutine to reset the interrupt request. The  $\overline{\text{INT}}$  pin may also be tested using the conditional jump instruction JNI. This instruction may be used to detect the presence of a pending interrupt before interrupts are enabled. If interrupt is left disabled,  $\overline{\text{INT}}$  may be used as another test input like T0 and T1.

## 2.1.10 Timer/Counter

The 8048H contains a counter to aid the user in counting external events and generating accurate time delays without placing a burden on the processor for these functions. In both modes the counter operation is the same, the only difference being the source of the input to the counter. The timer/event counter is shown in Figure 2-9.

### COUNTER

The 8-bit binary counter is presetable and readable with two MOV instructions which transfer the contents of the accumulator to the counter and vice versa. The counter content may be affected by Reset and should be initialized by software. The counter is stopped by a Reset or STOP TCNT instruction and remains stopped until started as a timer by a START T instruction or as an event counter by a START CNT instruction. Once started the counter will increment to this maximum count (FF) and overflow to zero continuing its count until stopped by a STOP TCNT instruction or Reset.

The increment from maximum count to zero (overflow) results in the setting of an overflow flag flip-flop and in the generation of an interrupt request. The state of the overflow flag is testable with the conditional jump instruction JTF. The flag is reset by executing a JTF or by Reset. The interrupt request is stored in a latch and then ORed with the external interrupt input INT. The timer interrupt may be enabled or disabled independently of external interrupt by the EN TCNTI and DIS TCNTI instructions. If enabled, the counter overflow will cause a subroutine call to location 7 where the timer or counter service routine may be stored.



## SINGLE COMPONENT MCS-48 SYSTEM

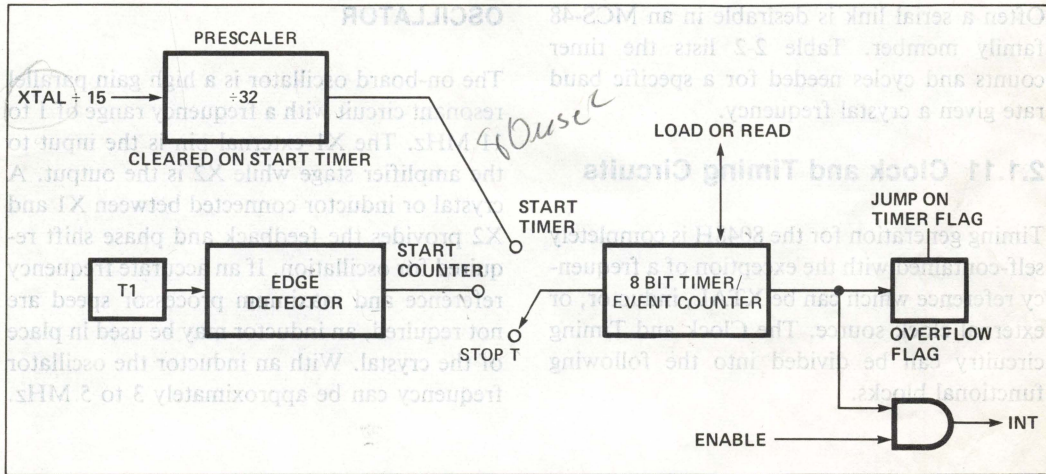


Figure 2-9. Timer/Event Counter

If timer and external interrupts occur simultaneously, the external source will be recognized and the Call will be to location 3. Since the timer interrupt is latched it will remain pending until the external device is serviced and immediately be recognized upon return from the service routine. The pending timer interrupt is reset by the Call to location 7 or may be removed by executing a DIS TCNTI instruction.

### AS AN EVENT COUNTER

Execution of a START CNT instruction connects the T1 input pin to the counter input and enables the counter. The T1 input is sampled at the beginning of state 3 or in later MCS-48 devices in state time 4. Subsequent high to low transitions on T1 will cause the counter to increment. T1 must be held low for at least 1 machine cycle to insure it won't be missed. The maximum rate at which the counter may be incremented is once per three instruction cycles (every  $5.7 \mu\text{sec}$  when using an 8 MHz crystal) — there is no minimum frequency. T1 input

must remain high for at least 1/5 machine cycle after each transition.

### AS A TIMER

Execution of a START T instruction connects an internal clock to the counter input and enables the counter. The internal clock is derived by passing the basic machine cycle clock through a  $\div 32$  prescaler. The prescaler is reset during the START T instruction. The resulting clock increments the counter every 32 machine cycles. Various delays from 1 to 256 counts can be obtained by presetting the counter and detecting overflow. Times longer than 256 counts may be achieved by accumulating multiple overflows in a register under software control. For time resolution less than 1 count an external clock can be applied to the T1 input and the counter operated in the event counter mode. ALE divided by 3 or more can serve as this external clock. Very small delays or "fine tuning" of larger delays can be easily accomplished by software delay loops.



## SINGLE COMPONENT MCS-48 SYSTEM

Often a serial link is desirable in an MCS-48 family member. Table 2-2 lists the timer counts and cycles needed for a specific baud rate given a crystal frequency.

### 2.1.11 Clock and Timing Circuits

Timing generation for the 8048H is completely self-contained with the exception of a frequency reference which can be XTAL, inductor, or external clock source. The Clock and Timing circuitry can be divided into the following functional blocks.

### OSCILLATOR

The on-board oscillator is a high gain parallel resonant circuit with a frequency range of 1 to 11 MHz. The X1 external pin is the input to the amplifier stage while X2 is the output. A crystal or inductor connected between X1 and X2 provides the feedback and phase shift required for oscillation. If an accurate frequency reference and maximum processor speed are not required, an inductor may be used in place of the crystal. With an inductor the oscillator frequency can be approximately 3 to 5 MHz.

**Table 2-2. Baud Rate Generation**

| Frequency<br>(MHz) |  | T <sub>cy</sub>                          | T0 Prr(1/5 T <sub>cy</sub> )             | Timer Prescaler<br>(32 T <sub>cy</sub> )  |
|--------------------|--|--|--|---|
| 4                  |  | 3.75 $\mu$ s                             | 750 ns                                   | 120 $\mu$ s                               |
| 6                  |  | 2.50 $\mu$ s                             | 500 ns                                   | 80 $\mu$ s                                |
| 8                  |  | 1.88 $\mu$ s                             | 375 ns                                   | 60.2 $\mu$ s                              |
| 11                 |  | 1.36 $\mu$ s                             | 275 ns                                   | 43.5 $\mu$ s                              |
| Baud Rate          | 4 MHz<br>Timer Counts +<br>Instr. Cycles | 6 MHz<br>Timer Counts +<br>Instr. Cycles | 8 MHz<br>Timer Counts +<br>Instr. Cycles | 11 MHz<br>Timer Counts +<br>Instr. Cycles |
| 110                | 75 + 24 Cycles<br>.01% Error             | 113 + 20 Cycles<br>.01% Error            | 151 + 3 Cycles<br>.01% Error             | 208 + 28 Cycles<br>.01% Error             |
| 300                | 27 + 24 Cycles<br>.1% Error              | 41 + 21 Cycles<br>.03% Error             | 55 + 13 Cycles<br>.01% Error             | 76 + 18 Cycles<br>.04% Error              |
| 1200               | 6 + 30 Cycles<br>.1% Error               | 10 + 13 Cycles<br>.1% Error              | 13 + 27 Cycles<br>.06% Error             | 19 + 4 Cycles<br>.12% Error               |
| 1800               | 4 + 20 Cycles<br>.1% Error               | 6 + 30 Cycles<br>.1% Error               | 9 + 7 Cycles<br>.17% Error               | 12 + 24 Cycles<br>.12% Error              |
| 2400               | 3 + 15 Cycles<br>.1% Error               | 5 + 6 Cycles<br>.4% Error                | 6 + 24 Cycles<br>.29% Error              | 9 + 18 Cycles<br>.12% Error               |
| 4800               | 1 + 23 Cycles<br>1.0% Error              | 2 + 19 Cycles<br>.4% Error               | 3 + 14 Cycles<br>.74% Error              | 4 + 25 Cycles<br>.12% Error               |



1.496



### Figure 2-10. MCS-48 Timing Generation and Cycle Timing

## STATE COUNTER

## CYCLE COUNTER

CLK is then divided by 5 in the Cycle Counter to provide a clock which defines a machine cycle consisting of 5 machine states as shown in Figure 2-10. Figure 2-11 shows the different internal operations as divided into the machine states. This clock is called Address Latch Enable (ALE) because of its function in MCS-48 systems with external memory. It is provided continuously on the ALE output pin.

### 2.1.12 Reset

The reset input provides a means for initialization for the processor. This Schmitt-trigger i



| INSTRUCTION             | CYCLE 1           |                           |                        |                     |                         | CYCLE 2              |               |                           |                  |    |
|-------------------------|-------------------|---------------------------|------------------------|---------------------|-------------------------|----------------------|---------------|---------------------------|------------------|----|
|                         | S1                | S2                        | S3                     | S4                  | S5                      | S1                   | S2            | S3                        | S4               | S5 |
| IN A, P                 | FETCH INSTRUCTION | INCREMENT PROGRAM COUNTER | —                      | * INCREMENT TIMER   | —                       | —                    | READ PORT     | —                         | * —              | —  |
| OUTL P, A               | FETCH INSTRUCTION | INCREMENT PROGRAM COUNTER | —                      | * INCREMENT TIMER   | OUTPUT TO PORT          | —                    | —             | —                         | * —              | —  |
| ANL P, :DATA            | FETCH INSTRUCTION | INCREMENT PROGRAM COUNTER | —                      | * INCREMENT TIMER   | READ PORT               | FETCH IMMEDIATE DATA | —             | INCREMENT PROGRAM COUNTER | * OUTPUT TO PORT | —  |
| ORL P, :DATA            | FETCH INSTRUCTION | INCREMENT PROGRAM COUNTER | —                      | * INCREMENT TIMER   | READ PORT               | FETCH IMMEDIATE DATA | —             | INCREMENT PROGRAM COUNTER | * OUTPUT TO PORT | —  |
| INS A, BUS              | FETCH INSTRUCTION | INCREMENT PROGRAM COUNTER | —                      | INCREMENT TIMER     | —                       | —                    | READ PORT     | —                         | * —              | —  |
| OUTL BUS, A             | FETCH INSTRUCTION | INCREMENT PROGRAM COUNTER | —                      | INCREMENT TIMER     | OUTPUT TO PORT          | —                    | —             | —                         | * —              | —  |
| ANL BUS, :DATA          | FETCH INSTRUCTION | INCREMENT PROGRAM COUNTER | —                      | * INCREMENT TIMER   | READ PORT               | FETCH IMMEDIATE DATA | —             | INCREMENT PROGRAM COUNTER | * OUTPUT TO PORT | —  |
| ORL BUS, :DATA          | FETCH INSTRUCTION | INCREMENT PROGRAM COUNTER | —                      | * INCREMENT TIMER   | READ PORT               | FETCH IMMEDIATE DATA | —             | INCREMENT PROGRAM COUNTER | * OUTPUT TO PORT | —  |
| MOVX @ R, A             | FETCH INSTRUCTION | INCREMENT PROGRAM COUNTER | OUTPUT RAM ADDRESS     | INCREMENT TIMER     | OUTPUT DATA TO RAM      | —                    | —             | —                         | * —              | —  |
| MOVX A, @R              | FETCH INSTRUCTION | INCREMENT PROGRAM COUNTER | OUTPUT RAM ADDRESS     | INCREMENT TIMER     | —                       | —                    | READ DATA     | —                         | * —              | —  |
| MOVD A, P <sub>i</sub>  | FETCH INSTRUCTION | INCREMENT PROGRAM COUNTER | OUTPUT OP CODE/ADDRESS | INCREMENT TIMER     | —                       | —                    | READ P2 LOWER | —                         | * —              | —  |
| MOVD P <sub>i</sub> , A | FETCH INSTRUCTION | INCREMENT PROGRAM COUNTER | OUTPUT OP CODE/ADDRESS | INCREMENT TIMER     | OUTPUT DATA TO P2 LOWER | —                    | —             | —                         | * —              | —  |
| ANLD P, A               | FETCH INSTRUCTION | INCREMENT PROGRAM COUNTER | OUTPUT OP CODE/ADDRESS | INCREMENT TIMER     | OUTPUT DATA             | —                    | —             | —                         | * —              | —  |
| ORLD P, A               | FETCH INSTRUCTION | INCREMENT PROGRAM COUNTER | OUTPUT OP CODE/ADDRESS | INCREMENT TIMER     | OUTPUT DATA             | —                    | —             | —                         | * —              | —  |
| J (CONDITIONAL)         | FETCH INSTRUCTION | INCREMENT PROGRAM COUNTER | SAMPLE CONDITION       | * INCREMENT TIMER   | —                       | FETCH IMMEDIATE DATA | —             | UPDATE PROGRAM COUNTER    | * —              | —  |
| STRT T                  | FETCH INSTRUCTION | INCREMENT PROGRAM COUNTER | —                      | * —                 | START COUNTER           | —                    | —             | —                         | —                | —  |
| STOP TCNT               | FETCH INSTRUCTION | INCREMENT PROGRAM COUNTER | —                      | * —                 | STOP COUNTER            | —                    | —             | —                         | —                | —  |
| ENI                     | FETCH INSTRUCTION | INCREMENT PROGRAM COUNTER | —                      | * ENABLE INTERRUPT  | —                       | —                    | —             | —                         | —                | —  |
| DIS I                   | FETCH INSTRUCTION | INCREMENT PROGRAM COUNTER | —                      | * DISABLE INTERRUPT | —                       | —                    | —             | —                         | —                | —  |
| ENTO CLK                | FETCH INSTRUCTION | INCREMENT PROGRAM COUNTER | —                      | * ENABLE CLOCK      | —                       | —                    | —             | —                         | —                | —  |

\*VALID INSTRUCTION ADDRESSES ARE OUTPUT AT THIS TIME IF EXTERNAL PROGRAM MEMORY IS BEING ACCESSED.

(1) IN LATER MCS-48 DEVICES T1 IS SAMPLED IN S4.

SINGLE COMPONENT MCS-48 SYSTEM

Figure 2-11. 8048H/8049H Instruction Timing Diagram



## SINGLE COMPONENT MCS-48 SYSTEM

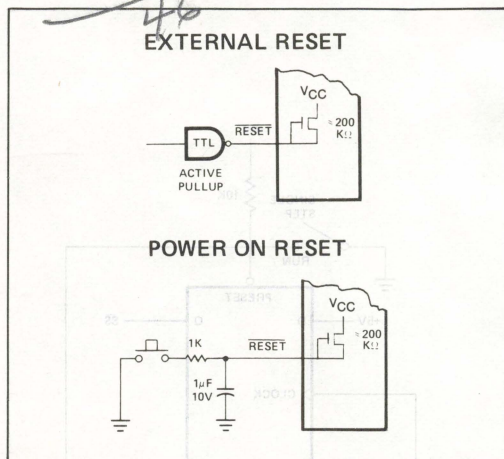


Figure 2-12

put has an internal pullup device which in combination with an external 1  $\mu$ F capacitor provides an internal reset pulse of sufficient length to guarantee all circuitry is reset, as shown in Figure 2-12. If the reset pulse is generated externally the RESET pin must be held low for at least 10 milliseconds after the power supply is within tolerance. Only 5 machine cycles ( $9.5 \mu\text{s}$  @ 8 MHz) are required if power is already on and the oscillator has stabilized.

Reset performs the following functions:

- 1) Sets program counter to zero.
- 2) Sets stack pointer to zero.
- 3) Selects register bank 0.
- 4) Selects memory bank 0.
- 5) Sets BUS to high impedance state (except when  $EA = 5V$ ).
- 6) Sets Ports 1 and 2 to input mode.
- 7) Disables interrupts (timer and external).
- 8) Stops timer.
- 9) Clears timer flag.
- 10) Clears F0 and F1.
- 11) Disables clock output from T0.

### 2.1.13 Single-Step

This feature, as pictured in Figure 2-13, provides the user with a debug capability in that the processor can be stepped through the program one instruction at a time. While stopped, the address of the next instruction to be fetched is available concurrently on BUS and the lower half of Port 2. The user can therefore follow the program through each of the instruction steps. A timing diagram, showing the interaction between output ALE and input  $\overline{SS}$ , is shown. The BUS buffer contents are lost during single step; however, a latch may be added to re-establish the lost I/O capability if needed. Data is valid at the leading edge of ALE.

### TIMING

The 8048H operates in a single-step mode as follows:

- 1) The processor is requested to stop by applying a low level on  $\overline{SS}$ .
- 2) The processor responds by stopping during the address fetch portion of the next instruction. If a double cycle instruction is in progress when the single step command is received, both cycles will be completed before stopping.
- 3) The processor acknowledges it has entered the stopped state by raising ALE high. In this state (which can be maintained indefinitely) the address of the next instruction to be fetched is present on BUS and the lower half of port 2.
- 4)  $\overline{SS}$  is then raised high to bring the processor out of the stopped mode allowing it to fetch the next instruction. The exit from stop is indicated by the processor bringing ALE low.
- 5) To stop the processor at the next instruction...



# SINGLE COMPONENT MCS-48 SYSTEM

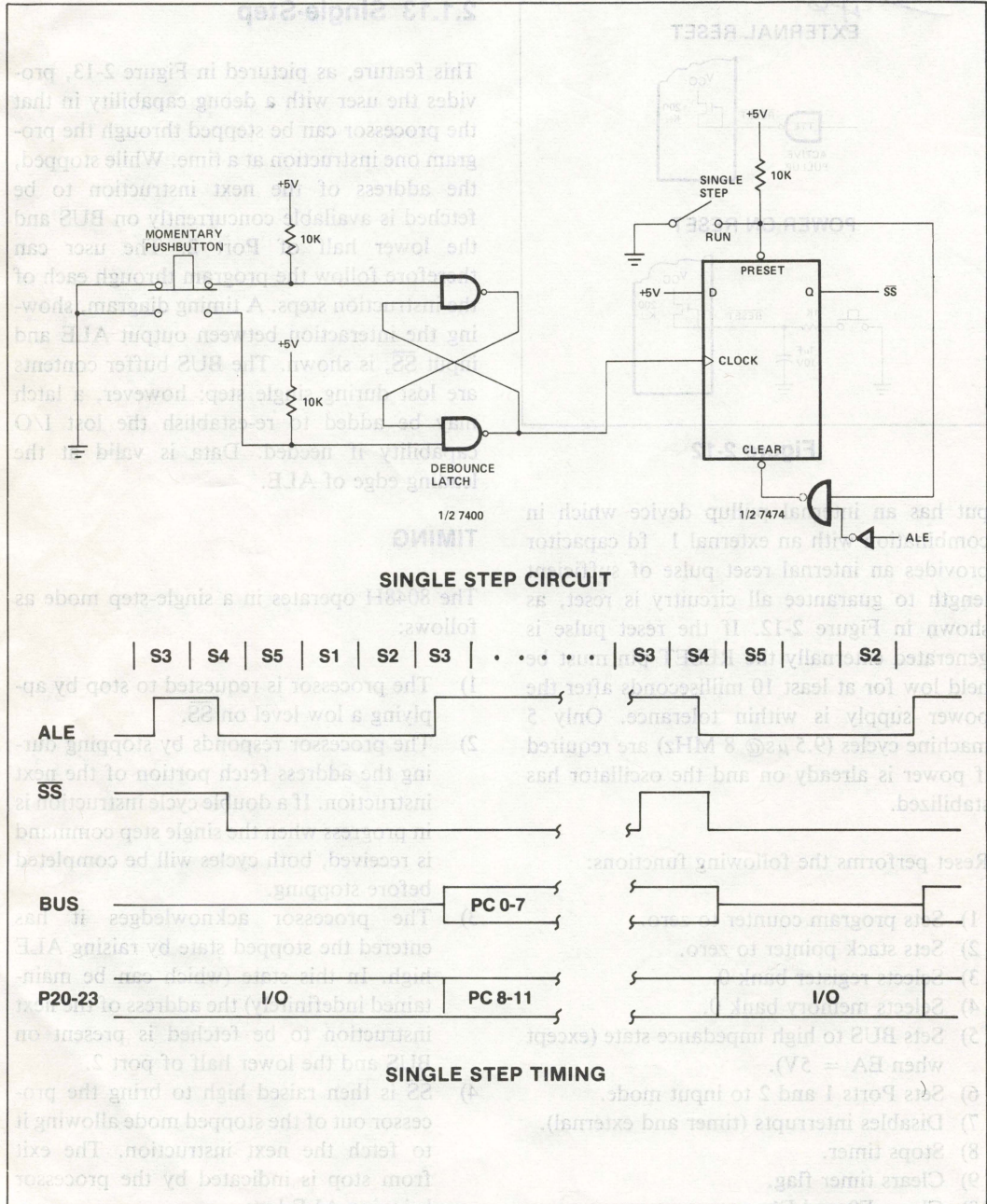


Figure 2-13. Single Step Operation



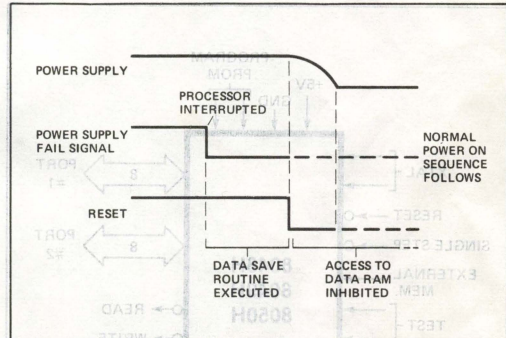
tion  $\overline{SS}$  must be brought low again soon after ALE goes low. If  $\overline{SS}$  is left high the processor remains in a "Run" mode.

A diagram for implementing the single-step function of the 8748 is shown in Figure 2-13. A D-type flip-flop with preset and clear is used to generate  $\overline{SS}$ . In the run mode  $\overline{SS}$  is held high by keeping the flip-flop preset (preset has precedence over the clear input). To enter single step, preset is removed allowing ALE to bring  $\overline{SS}$  low via the clear input. ALE should be buffered since the clear input of an SN7474 is the equivalent of 3 TTL loads. The processor is now in the stopped state. The next instruction is initiated by clocking a "1" into the flip-flop. This "1" will not appear on  $\overline{SS}$  unless ALE is high removing clear from the flip-flop. In response to  $\overline{SS}$  going high the processor begins an instruction fetch which brings ALE low resetting  $\overline{SS}$  through the clear input and causing the processor to again enter the stopped state.

## 2.1.14 Power Down Mode (8048H, 8049H, 8050H, 8039HL, 8035HL, 8040AHL)

Extra circuitry has been added to the 8048H/8049H/8050 ROM version to allow power to be removed from all but the data RAM array for low power standby operation. In the power down mode the contents of data RAM can be maintained while drawing typically 10% to 15% of normal operating power requirements.

$V_{CC}$  serves as the 5V supply pin for the bulk of circuitry while the  $V_{DD}$  pin supplies only the RAM array. In normal operation both pins are a 5V while in standby,  $V_{CC}$  is at ground and  $V_{DD}$  is maintained at its standby value. Applying Reset to the processor through the RESET



**Figure 2-14. Power Down Sequence**

pin inhibits any access to the RAM by the processor and guarantees that RAM cannot be inadvertently altered as power is removed from  $V_{CC}$ .

A typical power down sequence (Figure 2-14) occurs as follows:

- 1) Imminent power supply failure is detected by user defined circuitry. Signal must be early enough to allow 8048H to save all necessary data before  $V_{CC}$  falls below normal operating limits.
- 2) Power fail signal is used to interrupt processor and vector it to a power fail service routine.
- 3) Power fail routine saves all important data and machine status in the internal data RAM array. Routine may also initiate transfer of backup supply to the  $V_{DD}$  pin and indicate to external circuitry that power fail routine is complete.
- 4) Reset is applied to guarantee data will not be altered as the power supply falls out of limits. Reset must be held low until  $V_{CC}$  is at ground level.

Recovery from the Power Down mode can occur as any other power-on sequence with an



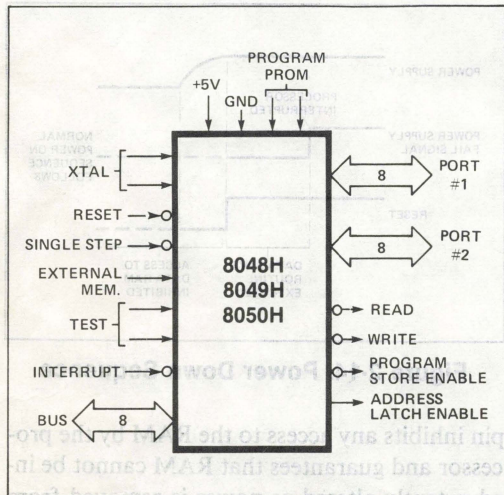


Figure 2-15. 8048H and 8049H  
Logic Symbol

external capacitor on the Reset input providing the necessary delay. See the previous section on Reset.

### 2.1.15 External Access Mode

Normally the first 1K (8048H), 2K (8049H), or 4K (8050H) words of program memory are automatically fetched from internal ROM or EPROM. The EA input pin however allows the user to effectively disable internal program memory by forcing all program memory fetches to reference external memory. The following chapter explains how access to external program memory is accomplished.

The External Access mode is very useful in system test and debug because it allows the user to disable his internal applications program and substitute an external program of his choice—a diagnostic routine for instance. In addition, the section on Test and Debug explains how internal program memory can be read externally, independent of the processor.

A “1” level on EA initiates the external access mode. For proper operation, Reset should be applied while the EA input is changed.

## 2.2 PIN DESCRIPTION

The MCS-48 processors (except 8021H and 8020H) are packaged in 40 pin Dual In-Line Packages (DIP's). Table 2-3 is a summary of the functions of each pin. Figure 2-15 is the logic symbol for the 8048H product family. Where it exists, the second paragraph describes each pin's function in an expanded MCS-48 system. Unless otherwise specified, each input is TTL compatible and each output will drive one standard TTL load.

## 2.3 PROGRAMMING, VERIFYING AND ERASING EPROM

The internal Program Memory of the 8748 and the 8749H may be erased and reprogrammed by the user as explained in the following sections. See also the 8748 and 8749H data sheets.

### 2.3.1 Programming/Verification

In brief, the programming process consists of: activating the program mode, applying an address, latching the address, applying data, and applying a programming pulse. This programming algorithm applies to both the 8748 and 8749H. The only differences between the two parts during programming are the lower voltages on the  $V_{DD}$ , EA and PROG pins of the 8749H (see Figures 2-16 and 2-17) and tying P10 and P11 to ground. Each word is programmed completely before moving on to the next and is followed by a verification step. The following is a list of the pins used for programming and a description of their functions:



# 31 32 8749H **SINGLE COMPONENT MCS-48 SYSTEM**

**Table 2-3**

| Designation         | Pin Number*    | Function  |
|---------------------|----------------|---|
| V <sub>SS</sub>     | 20             | Circuit GND potential   |
| V <sub>DD</sub>     | 26             | Programming power supply; +25V during program for the 8748 and 21V during program for the 8748H/8749H; +5V during operation for both ROM and PROM. Low power standby pin in 8048H and 8049H/8050H ROM versions.   |
| V <sub>CC</sub>     | 40             | Main power supply; +5V during operation and 8748 and 8749H programming.   |
| PROG                | 25             | Program pulse; +23V input pin during 8748 programming and 18V input pin during 8748H/8749H programming. Output strobe for 8243 I/O expander.  |
| P10-P17<br>(Port 1) | 27-34          | 8-bit quasi-bidirectional port. (Internal Pullup $\approx 50K \Omega$ )   |
| P20-P27<br>(Port 2) | 21-24<br>35-38 | 8-bit quasi-bidirectional port. (Internal Pullup $\approx 50K \Omega$ )<br><br>P20-P23 contain the four high order program counter bits during an external program memory fetch and serve as a 4-bit I/O expander bus for 8243.   |
| D0-D7<br>(BUS)      | 12-19          | True bidirectional port which can be written or read synchronously using the RD, WR strobes. The port can also be statically latched.<br><br>Contains the 8 low order program counter bits during an external program memory fetch, and receives the addressed instruction under the control of PSEN. Also contains the address and data during an external RAM data store instruction, under control of ALE, RD, and WR. |
| T0                  | 1              | Input pin testable using the conditional transfer instructions JT0 and JNT0. T0 can be designated as a clock output using ENT0 CLK instruction. T0 is also used during programming.   |
| T1                  | 39             | Input pin testable using the JT1, and JNT1 instructions. Can be designated the event counter input using the STRT CNT instruction. (See Section 2.1.10)   |
| INT                 | 6              | Interrupt input. Initiates an interrupt if interrupt is enabled. Interrupt is disabled after a reset. (Active low)<br><br>Interrupt must remain low for at least 3 machine cycles to ensure proper operation.   |
| RD                  | 8              | Output strobe activated during a BUS read. Can be used to enable data onto the BUS from an external device. (Active low)<br><br>Used as a Read Strobe to External Data Memory.  |



## SINGLE COMPONENT MCS-48 SYSTEM

**Table 2-3 (Continued)**

| Designation | Pin Number | Function  |
|-------------|------------|---|
| RESET       | 4          | Input which is used to initialize the processor. Also used during PROM programming and verification. (Active low) (Internal pullup $\approx 200K\Omega$ )   |
| WR          | 10         | Output strobe during a BUS write. (Active low) Used as write strobe to external data memory.  |
| ALE         | 11         | Address Latch Enable. This signal occurs once during each cycle and is useful as a clock output.<br><br>The negative edge of ALE strobes address into external data and program memory.   |
| PSEN        | 9          | Program Store Enable. This output occurs only during a fetch to external program memory. (Active low)   |
| SS          | 5          | Single step input can be used in conjunction with ALE to "single step" the processor through each instruction. (Active low) (Internal pullup $\approx 300K\Omega$ )   |
| EA          | 7          | External Access input which forces all program memory fetches to reference external memory. Useful for emulation and debug, and essential for testing and program verification. (Active high) 23V for 8748 Program Verification and 18V for 8748H/8749H verification (Internal pullup $\approx 10M\Omega$ on 8048H/8049H/8035HL/8039HL/8050H/8040H) |
| XTAL1       | 2          | One side of crystal input for internal oscillator. Also input for external source.  |
| XTAL2       | 3          | Other side of crystal/external source input.  |

\*Unless otherwise stated, inputs do not have internal pullup resistors. 8048H, 8748, 8049H, 8050H, 8040H

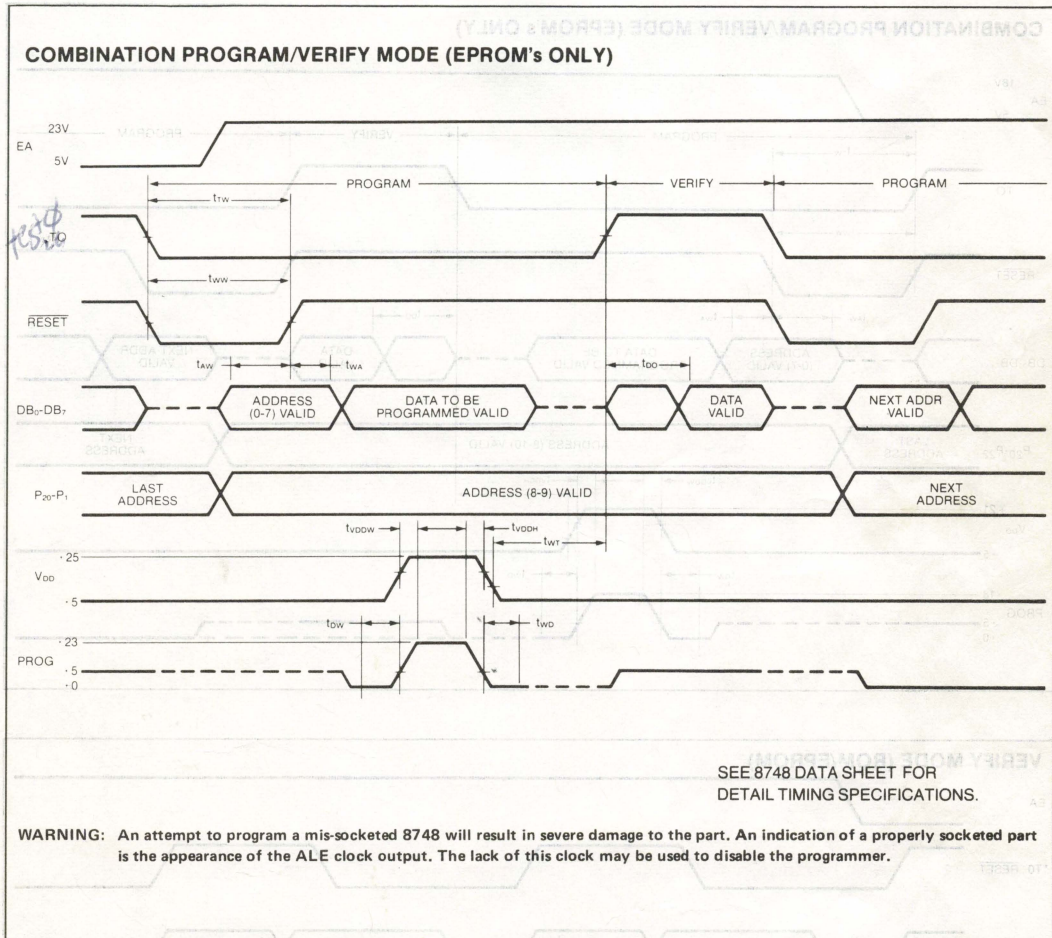
| Pin    | Function   | Pin  | Function                    |
|--------|--|--|-----------------------------|
| XTAL 1 | Clock Input (1 to 3MHz)                          | P20-2  | Address Input for 8749H     |
| Reset  | Initialization and Address Latching              | V <sub>DD</sub>  | Programming Power Supply    |
| Test 0 | Selection of Program (0V) or Verify (5V) Mode    | PROG   | Program Pulse Input         |
| EA     | Activation of Program/Verify Modes               | P10-P11  | Tied to ground (8749H only) |
| BUS    | Address and Data Input Data Output During Verify | <b>8748 AND 8749H ERASURE CHARACTERISTICS</b><br><br>The erasure characteristics of the 8748 and 8749H are such that erasure begins to occur |                             |
| P20-1  | Address Input for 8748                           |  |                             |



## SINGLE COMPONENT MCS-48 SYSTEM

### WAVEFORMS

WAVEFORMS



**Figure 2-16. Programming/Verify Sequence for the 8748**

when exposed to light with wavelengths shorter than approximately 4000 Angstroms ( $\text{\AA}$ ). It should be noted that sunlight and certain types of fluorescent lamps have wavelengths in the 3000-4000 $\text{\AA}$  range. Data show that constant exposure to room level fluorescent lighting could erase the typical 8748 and 8749H in approximately 3 years while it would take approximately 1 week to

cause erasure when exposed to direct sunlight. If the 8748 or 8749H is to be exposed to these types of lighting conditions for extended periods of time, opaque labels should be placed over the 8748 window to prevent unintentional erasure.

When erased, bits of the 8748 and 8749H Program Memory are in the logic "0" state.



# SINGLE COMPONENT MCS-48 SYSTEM

## WAVEFORMS

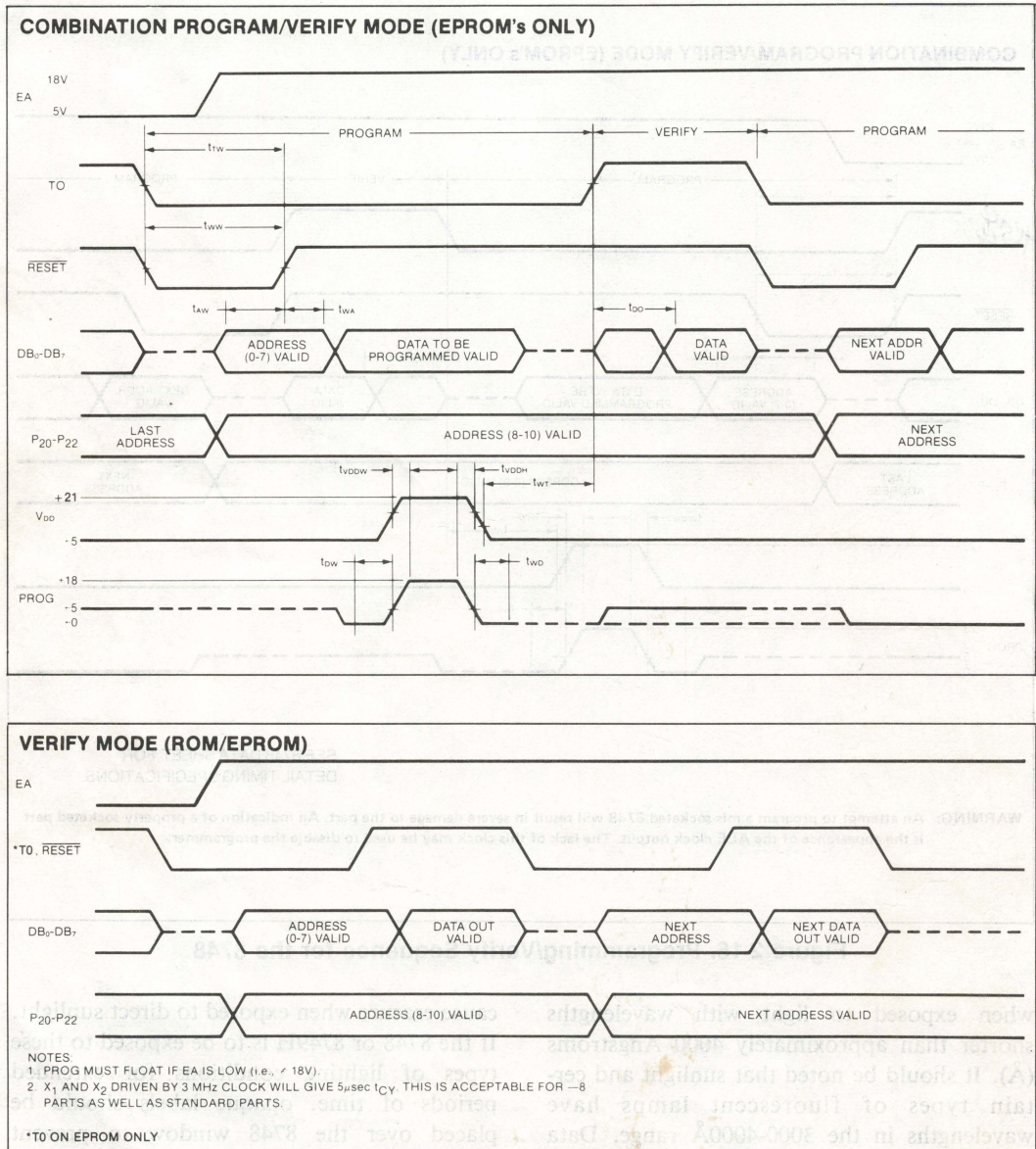


Figure 2-17. Program/Verify Sequence for 8749H



The recommended erasure procedure for the 8748 and 8749H is exposure to shortwave ultraviolet light which has a wavelength of 2537 Angstroms ( $\text{\AA}$ ). The integrated dose (i.e., UV intensity X exposure time) for erasure should be a minimum of  $15\text{W-sec/cm}^2$ . The erasure time with this dosage is approximately 15 to 20 minutes using an ultraviolet lamp with a  $12000\mu\text{W/cm}^2$  power rating. The 8748 and 8749H should be placed within one inch from the lamp tubes during erasure. Some lamps have a filter in their tubes and this filter should be removed before erasure.

The Program/Verify sequence is:

- 1)  $V_{DD} = 5\text{v}$ , Clock applied or internal oscillator operating,  $\text{Reset} = 0\text{v}$ , Test 0 = 5v, EA = 5v, BUS and PROG floating. P10 and P11 tied to ground (8749H only)
- 2) Insert 8748 or 8749H in programming socket
- 3) Test 0 = 0v (Select Program Mode)
- 4) EA = 23V Activate Program Mode for 8748)  
EA = 18v (Activate Program Mode for 8748H/8749H)
- 5) Address applied to BUS and P20-1 (2)
- 6)  $\text{Reset} = 5\text{v}$  (Latch Address)
- 7) Data applied to BUS
- 8)  $V_{DD} = 25\text{v}$  (Programming Power for 8748)  
 $V_{DD} = 21\text{v}$  (Programming Power for 8748H/8749H)
- 9) PROG = 0v followed by one 50ms pulse to 23V for the 8748.  
PROG = 0v followed by one 50 msec to 18v for the 8748H/8749H.
- 10)  $V_{DD} = 5\text{v}$
- 11) TEST 0 = 5v (Verify Mode)
- 12) Read and Verify Data on BUS
- 13) TEST 0 = 0v
- 14)  $\text{Reset} = 0\text{v}$  and repeat from Step 5

- 15) Programmer should be at conditions of Step 1 when 8748 or 8749H is removed from socket.

## 2.4 READING INTERNAL PROGRAM MEMORY

Just as the processor may be isolated from internal program memory using EA, program memory can be read independent of the processor using the verification mode described in the previous section, Programming/Verification. Figure 2-18 shows the waveforms for the reading of internal program memory.

The processor is placed in the READ mode by applying a high voltage (+23V for the 8748, +18V for the 8749H and +12V for the 8048H/8049H/8050H) to the EA pin and +5V to the T0 (8748 and 8749H only) input pin.  $\text{RESET}$  must be at 0V when voltage is applied to EA. The address of the location to be read is then applied to the BUS and Port 2. All twelve address bits must be driven. The address is latched by a "0" to "1" transition on  $\text{RESET}$  and a high level on  $\text{RESET}$  causes the contents of the program memory location addressed to appear on the eight lines of BUS.  $\text{RESET}$  must be brought back to 0V before leaving the READ mode.

## 2.5 8020H/8021H FUNCTIONAL SPECIFICATIONS

The following is a functional description of the major elements of the 8020H and 8021H. It is important to understand at the outset, however, that the 8020H is itself an 8021H but in a 20-pin DIP. This significant pin reduction is accomplished by bonding out only 13 of the 8021H's 21 I/O lines: Port 00-07 and Port 13-17.



## SINGLE COMPONENT MCS-48 SYSTEM

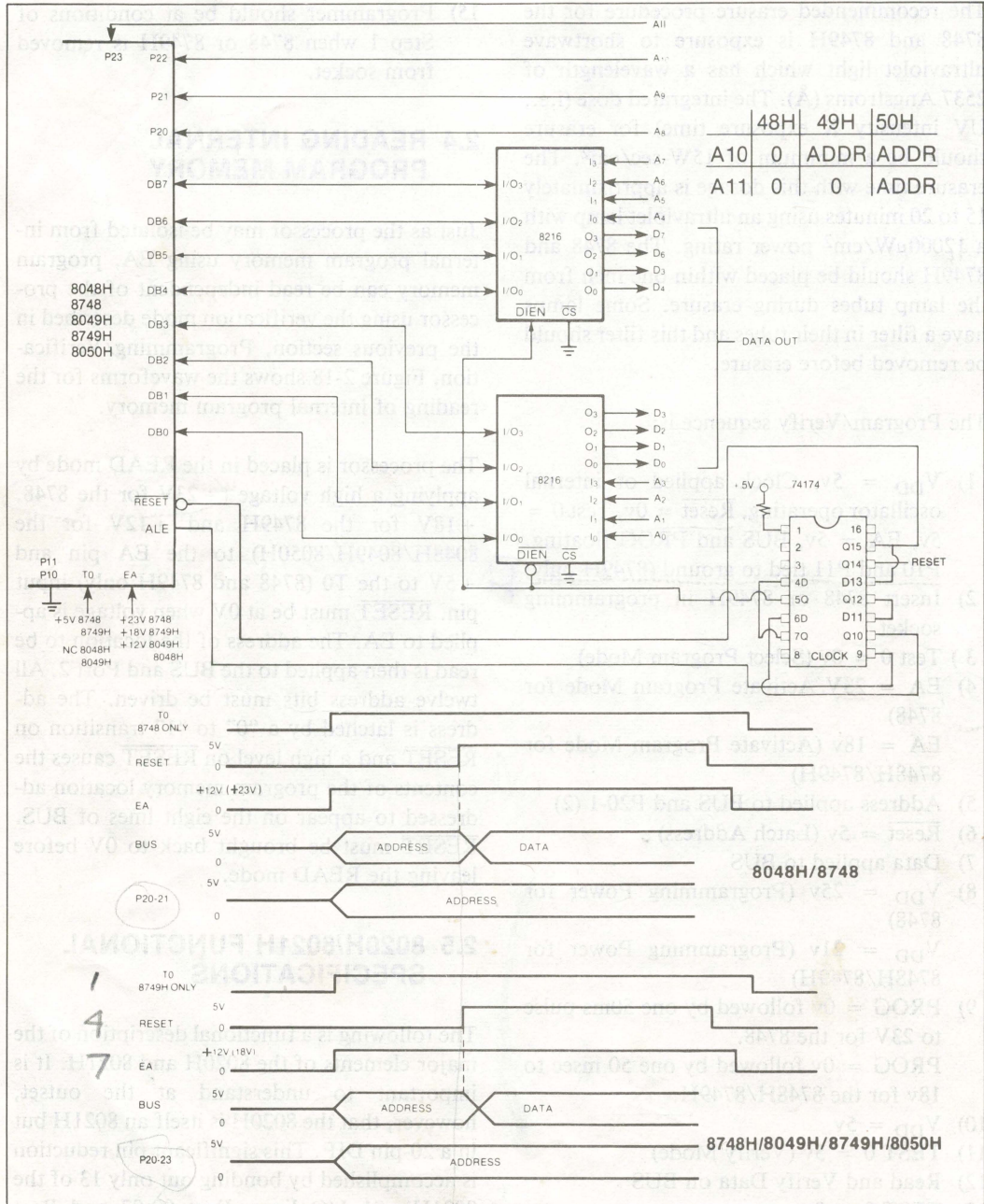


Figure 2-18. Reading Internal Program Memory



## SINGLE COMPONENT MCS-48 SYSTEM

### 2.5.1 Program Memory

The 8020H/8021H contains 1K x 8 of mask programmable ROM. No external ROM expansion capability is provided.

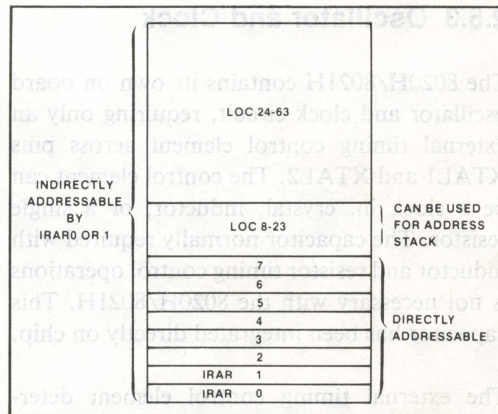
### 2.5.2 Data Memory

A 64 x 8 dynamic RAM is located on chip for data storage. All locations are indirectly addressable and eight designated locations are directly addressable. Also, included in the memory is the address stack, addressed by a 3-bit stack pointer.

Memory is organized as shown in Figure 2-19. The least significant 8 addresses, 0-7, are directly addressable by any of the 11 direct register instructions. The locations are readily accessible for a variety of operations with the least number of instruction bytes required for their manipulation.

Registers 0 and 1 have another function, in that they can be used to indirectly address all locations in memory, using the indirect register instructions. These indirect RAM address registers, IRAR's, are especially useful for repetitive-type operations on adjacent memory locations. The indirect register instruction specifies which IRAR to use, and the contents of the IRAR is used to address a location in RAM. The contents of the addressed location is used during the execution of the instruction and may be modified. A value larger than 63 should not be present in the IRAR when selected by an indirect register instruction. IRAR's may point to addresses 0-7, if desired.

Locations 8-23 may be used as the address stack. The address stack enables the processor to keep track of the return addresses generated from CALL instructions. A 3-bit stack pointer



**Figure 2-19. Internal RAM Organization**

(SP) supplies the address of the locations to be loaded with the next return address generated. The SP to this pushdown stack is incremented by one after a return address is stored, and decremented by one before an address is fetched during a RET. The unincremented program counter address is stored in the address stack. Before loading the program counter during a return from the subroutine (RET), the stack contents are incremented. A total of 8 levels of nesting is possible. The SP is initialized to location 8 upon RESET. Since each address is 10-bits long, two bytes must be used to store a single address. The SP is incremented and decremented by one, but each increment or decrement moves the address pointed to by two. Therefore, only even numbered addresses are pointed to. If a particular application does not require 8 levels of nesting, the unused portion of the stack may be used as any other indirectly addressable scratchpad location. For example, if only 3 levels of subroutine nesting are used, then only locations 8-13 need be reserved for the address stack, and locations 14-63 can be used for data storage.



## SINGLE COMPONENT MCS-48 SYSTEM

### 2.5.3 Oscillator and Clock

The 8020H/8021H contains its own on board oscillator and clock circuit, requiring only an external timing control element across pins XTAL1 and XTAL2. The control element can be a clock in, crystal, inductor, or a single resistor. The capacitor normally required with inductor and resistor timing control operations is not necessary with the 8020H/8021H. This capacitor has been integrated directly on chip.

The external timing control element determines all internal time divisions. An instruction cycle consists of 10 states, and each state is a time division of 3 oscillator periods (see Figure 2-20). Therefore, to obtain a 10  $\mu$ sec instruction cycle, a 3 MHz oscillator frequency is required. This may be obtained with a direct 3 MHz clock in, a 3 MHz crystal, or a 330  $\mu$ H inductor. A 15K resistor will also yield an oscillator frequency in the range of 3 to 3.6 MHz. Note, however, that the required inductance and resistance may vary and should be adjusted as necessary. The resistance and inductance are inversely proportional to the generated oscillator frequency.

The 8020H/8021H utilizes dynamic RAM and

certain other dynamic logic. Due to the clocking required with dynamic circuits, the oscillator frequency must be equal to or greater than 600 kHz, or improper operation may occur.

### 2.5.4 Timer/Event Counter

The 8020H/8021H has internal timer/event counter circuits that can monitor elapsed time or count external events that occur during program execution. The circuit has an 8-bit binary up-counter that is presetable and readable with two MOV instructions. These instructions transfer the contents of the accumulator to the counter and vice versa. The counter is cleared by Reset and can be set by the MOV T,A instruction. The counter is stopped by a RESET or STOP TCNT instruction and remains stopped until started as a timer by a STRT T instruction or as an event counter by a STRT CNT instruction. Once started, the counter increments to its maximum count (FF), and overflows to zero. The count continues until stopped by a STOP TCNT instruction or RESET. The increment from maximum count to zero (overflow) sets an overflow flag. The state of the overflow flag is testable with the conditional jump instruction JTF. The flag is

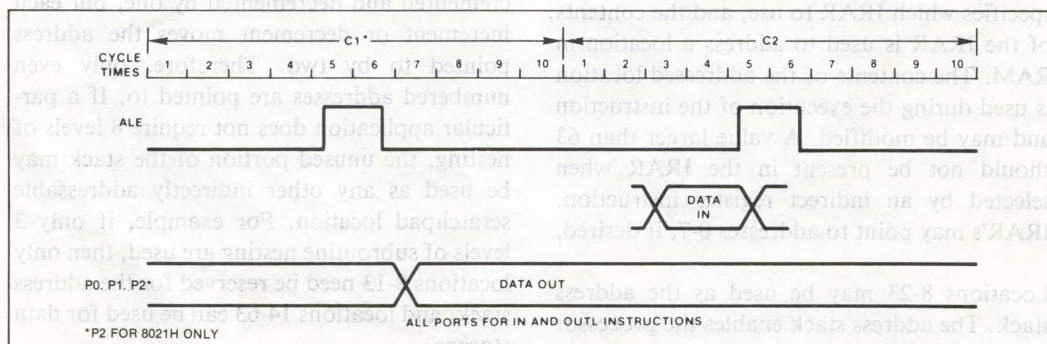


Figure 2-20. 8020H/8021H Timing Diagram



## SINGLE COMPONENT MCS-48 SYSTEM

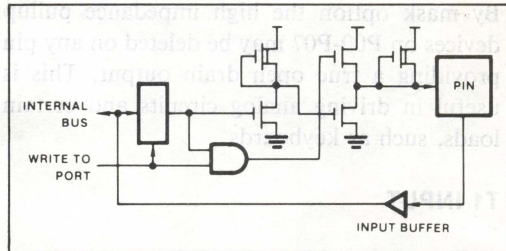
reset by JTF but not by executing a RESET, unlike the 8748.

At the STRT T command an internal prescaler is zeroed and thereafter increments once each 30 input clocks (once each single cycle instruction, twice each double cycle instruction). The prescaler is a divide by 32. At the (11111) to (00000) transition the timer is incremented. The timer is 8-bits and an overflow (FFH) to (00H) timer flag is set. A conditional branch instruction (JTF) is available for testing this flag, the flag being reset each test. Total count capacity for the timer is  $2^8 \times 2^5 = 8192$  or 81.9 msec at a  $10 \mu\text{sec}$  cycle time. Contents of the timer can be moved to the accumulator by the MOV A,T instruction without disturbing the counting process.

Execution of a START CNT instruction connects the T1 input pin to the counter input and enables the counter. The T1 input is sampled at the beginning of state 3. Subsequent high to low transitions on T1 will cause the counter to increment. T1 must be held for at least  $1T_{cy}$  to ensure it will not be missed. The maximum rate at which the counter may be incremented is once per three instruction cycles (every  $30 \mu\text{s}$  for a 3 MHz oscillator) — there is no minimum rate. T1 input must remain stable for at least  $1/5T_{cy}$  after each transition.

### 2.5.5 Input/Output Capabilities

The 8020H/8021H I/O configurations are highly flexible. A number of different configurations are possible, tailoring an 8020H/8021H to a given task. Other than the power supply and dedicated pins, all other pins can be used for input, output, or both, depending on the configuration.



**Figure 2-21. Quasi-Bidirectional Port Structure**

All ports are quasi-bidirectional to facilitate stand-alone use. A simplified schematic of the quasi-bidirectional interface is shown in Figure 2-21. This configuration allows buffered outputs, and also allows external input. Data written to these ports is statically latched and remains unchanged until rewritten. As input ports these lines are non-latching, i.e., inputs must be present until read by an input instruction. When writing a "0" or low value to these ports, the low impedance pulldown device sinks an external TTL load. When writing a "1", a large current is supplied through the low impedance pullup device to allow a fast data transfer. After a short time (less than one instruction cycle), the low impedance device is shut off and the high impedance pullup maintains the "1" level indefinitely. However, in this situation, an input device capable of overriding the small amount of sustaining current supplied by the pullup device can be read. (Alternatively, the data written can be read.) So, by writing a "1" to any particular pin, that pin can serve either as a true high-level latched output pin, or as just a pullup resistor on an input. This allows maximum user flexibility in selecting his input or latched output pins, with a minimum of external components.

Port 00-07 is also quasi-bidirectional, except there is no low impedance pullup device. As outputs, this port is essentially open drain.



By mask option the high impedance pullup devices on P00-P07 may be deleted on any pin providing a true open drain output. This is useful in driving analog circuits and certain loads, such as keyboards.

## T1 INPUT

The 8020H/8021H T1 input line can be used as an input for the following functions:

- Event Counter (external input)
- Test input for branch instructions
- Zero voltage crossing detection

The operation of T1 as an input to the Event Counter is described in the Timer/Event Counter section. When used as a test input, the JT1 and JNT1 instructions test for 1 and 0 levels, respectively.

The T1 pin can also be used to detect the zero crossing of slowly moving AC signals (60 Hz). The self-biasing circuit shown in Figure 2-22

permits the Test 1 input to detect when the input voltage crosses zero within  $\pm 5\%$ ; the voltage is then coupled through a  $1.0 \mu\text{f}$  capacitor. Maximum input voltage is 3V peak-to-peak. The zero cross detection is especially useful in SCR control of 60 Hz power and in developing time-of-day and other timing routines. As a ROM mask option there is a pullup device that is useful for switch contact input or standard TTL.

## HIGH CURRENT OUTPUTS

High current drive is desirable for minimizing external parts required to do high power control. P10 and P11, 8021H only, have been designated high drive outputs capable of sinking 7mA to  $V_{SS} + 2.5$  volts. (For clarity, this is 7mA to  $V_{SS}$  with a 2.5 volt drop across the buffer.) These pins may, of course, be paralleled for 14mA drive if the output logic states are always the same.

## EXPANDED I/O

The 8021H, not the 8020H however, can be used with the 8243 I/O expander chip, which provides additional I/O capability with a limited number of overhead pins. This chip has 4 directly addressable 4-bit ports. It connects to the PROG pin, which provides a clock, and pins P20-P23, which provide address and data. These ports can be written with a `MOVD P,A`; `ANLD P,A`; and `ORLD P,A` for Ports 4-7. A high to low transition on PROG signifies that address and control are available on P20-P23. The previous data on P20-P23 before an output expander instruction is lost. Therefore, when using an output expander P20-P23 are not useful for general input/output. Reading is via the `MOVD A,P`. This circuit configuration is shown in Figure

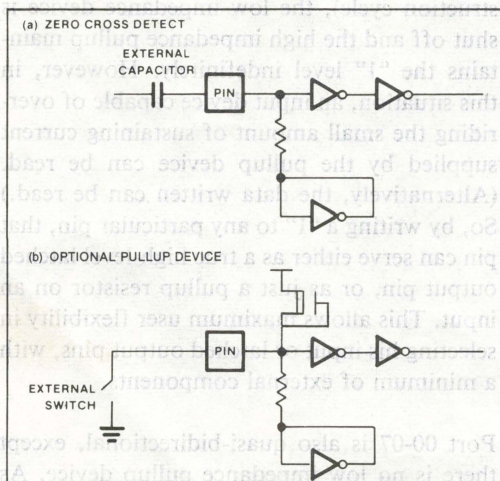


Figure 2-22. Test 1 Pin



## SINGLE COMPONENT MCS-48 SYSTEM

2-23. The timing diagram is shown in Figure 2-24.

Standard TTL can also be used to expand the number of I/O lines on the 8021H as well as the 8020H.

### 2.5.6 CPU

The 8020H/8021H CPU has arithmetic and logical capability. A wide variety of arithmetic and logic instructions may be exercised, which affect the contents of the accumulator and/or

direct or indirect scratchpad locations. Provisions have been made for simplified BCD arithmetic capability using the DAA, SWAP A and XCHD instructions. In addition, MOVP A,@A allows table lookup for display formatting and constants. The conditional branch logic within the processor enables several conditions internal and external to the processor to be tested by the users program. Use the conditional jump instructions with the tests listed below to effect a change in the program execution sequence:

| Test                | Jump Condition     | Jump Instructions |
|---------------------|--------------------|-------------------|
| Accumulator         | $A = 0$ $A \neq 0$ | JZ JNZ            |
| Carry Flag          | 0 1                | JC                |
| Timer Overflow Flag | — 1                | JTF               |
| Test Input-T1       | 0 1                | JNT1, JT1         |

### 2.5.7 Reset

The reset input provides a means for initialization for the processor. This Schmitt-trigger input has an internal pulldown device which in combination with an external  $1 \mu\text{f}$  capacitor provides an internal reset pulse of sufficient length to guarantee all circuitry is reset, as shown in Figure 2-25. If the reset pulse is

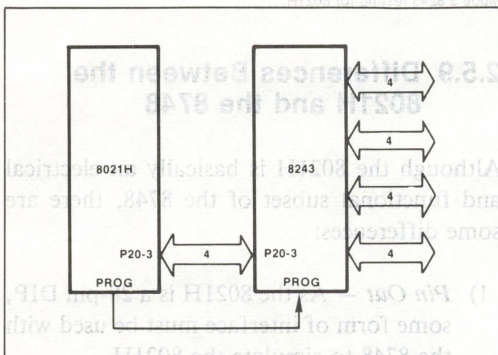


Figure 2-23. I/O Expander Interface

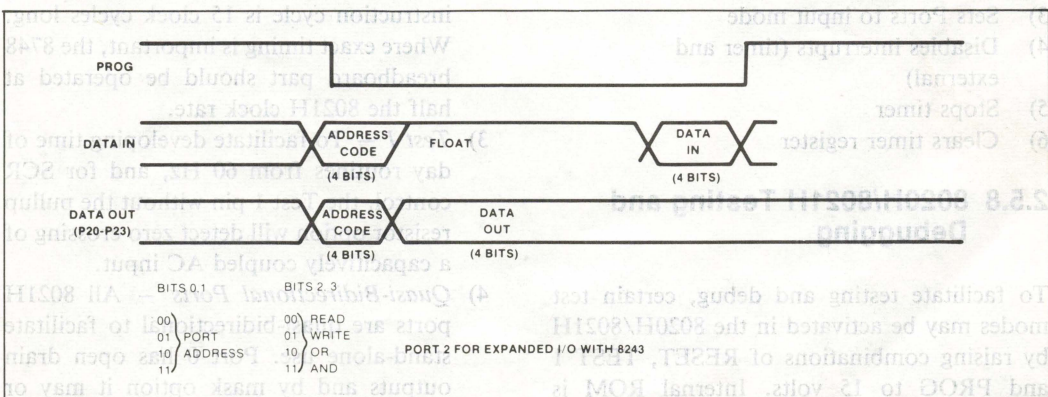
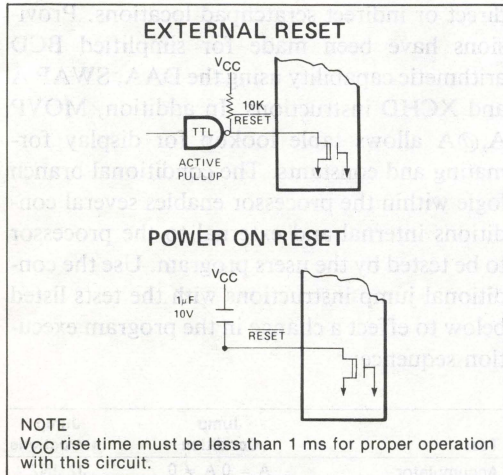


Figure 2-24. Expanded I/O Timing Diagram



## SINGLE COMPONENT MCS-48 SYSTEM



**Figure 2-25.**

generated externally, the RESET pin must be held above 3.8V for at least 10 milliseconds after the power supply is within tolerance. Only 3 machine cycles ( $2.5 \mu s$  @ 3.6 MHz) are required if power is already on and the oscillator has stabilized.

Reset performs the following functions:

- 1) Sets program counter to zero
- 2) Sets stack pointer to zero
- 3) Sets Ports to input mode
- 4) Disables interrupts (timer and external)
- 5) Stops timer
- 6) Clears timer register

### 2.5.8 8020H/8021H Testing and Debugging

To facilitate testing and debug, certain test modes may be activated in the 8020H/8021H by raising combinations of RESET, TEST 1 and PROG to 15 volts. Internal ROM is dumped out sequentially for verification. Ex-

ternal memory operation is used for CPU checkout.

| Reset | Prog | Test 1 | Case    | Function  |
|-------|------|--------|---------|---|
| 5V    | X    | X      |         | Power On Clear  |
| 0V    | X    | X      |         | Normal Operation  |
| 15V   | 15V  | 15V    | Mode 1a | On each cycle internal ROM is dumped to Port 0 — Sequentially after ALE leading edge.             |
| 15V   | 15V  |        | Mode 1b | On every TEST 1 falling edge the program counter increments, dumps internal ROM to Port 0.        |
| 0V    | 15V  | X      | Mode 2  | Chip will operate from external memory (one page) via Port 0. ALE strobes Address out, memory in. |
| 15V   | X    | X      | Mode 3  | Chip accepts op codes into Port 1. Allows Port 0 and 8243 testing.                                |

**NOTE**

X = Normal mode - between 0V and VCC  
Test 1 in Mode 1b should be limited to VCC  
Mode 3 8243 testing for 8021H.

### 2.5.9 Differences Between the 8021H and the 8748

Although the 8021H is basically an electrical and functional subset of the 8748, there are some differences:

- 1) *Pin Out* — As the 8021H is a 28-pin DIP, some form of interface must be used with the 8748 to simulate the 8021H.
- 2) *Instruction Time* — The 8021H instruction cycle is 30 clock cycles long, the 8748 instruction cycle is 15 clock cycles long. Where exact timing is important, the 8748 breadboard part should be operated at half the 8021H clock rate.
- 3) *Test 1* — To facilitate developing time of day routines from 60 Hz, and for SCR control, the Test 1 pin without the pullup resistor option will detect zero crossing of a capacitively coupled AC input.
- 4) *Quasi-Bidirectional Ports* — All 8021H ports are quasi-bidirectional to facilitate stand-alone use. Port 0 has open drain outputs and by mask option it may or may not have pullup devices.



- 5) *Oscillator* — The 8021H has an on-chip oscillator that is optimized for the single resistor mode. External connection will differ from the 8748.
- 6) *Dynamic RAM and Logic* — The 8021H utilizes dynamic RAM and some dynamic logic. Input clocking must be maintained above the minimum rate or improper operation may result.
- 7) *High Current Outputs* — High current drive is desirable for minimizing external parts required to do high power control. P10 and P11 have been designated high drive outputs capable of sinking 7mA at  $V_{SS} + 2.5$  volts. (For clarity, this is 7mA to  $V_{SS}$  with a 2.5 volt drop across the buffer.) These pins may, of course, be paralleled for 14mA drive if the output logic states are always the same.
- 8) *Timer/Counter* — The 8748 does not increment its timer in the second cycle of a 2-cycle instruction; the 8021H does.
- 9) *Reset* — Reset has been modified on the 8021H to active high; the 8748 is active low.
- 10) *Instruction Set* — The instructions below, which are found in the 8748, have been deleted from the 8021H instruction set:

## 2.6 8022 FUNCTIONAL SPECIFICATIONS

The 8022's architecture is based upon the 8021H, and many functions of the two parts are identical.

### 2.6.1 Program Memory

The 8022 program memory consists of 2048 words 8 bits wide which are addressed by the program counter. The memory is ROM which is mask programmable at the factory. No external ROM expansion capability is provided. There are three locations in program memory of special importance.

**Location 0:** Activating the RESET line of the processor causes the first instruction to be fetched from location 0.

**Location 3:** Activating the interrupt input line (T0) of the processor (if interrupt is enabled) causes a jump to subroutine.

**Location 7:** A timer/event counter interrupt resulting from a timer/counter overflow causes a jump to subroutine (if timer/counter interrupt is enabled).

| Data Moves | Registers    | Branch    | Timer             | Control  | Input/Output  |
|------------|--------------|-----------|-------------------|----------|---------------|
| MOV A,PSW  | DEC R        | JT0 addr  | EN TCNTI          | EN I     | ANL P,#data   |
| MOV PSW,A  |              | JNT0 addr | DIS TCNTI         | DIS I    | ORL P,#data   |
| MOVX A,@R  | <b>Flags</b> | JF0 addr  |                   | SEL RB0  | INS A,BUS*    |
| MOVX @R,A  | CLR F0       | JF1 addr  | <b>Subroutine</b> | SEL RB1  | OUTL BUS,A*   |
| MOVP3 A,@A | CPL F0       | JNI addr  |                   | SEL MB0  | ANL BUS,#data |
|            | CLR F1       | JBb addr  | RETR              | SEL MB1  | ORL BUS,#data |
|            | CPL F1       |           |                   | ENT0 CLK |               |

\*These instructions have been replaced in the 8021H by IN A,PO and OUTL PO,A, respectively.



## SINGLE COMPONENT MCS-48 SYSTEM

Therefore, the first instruction to be executed after initialization is stored in location 0. The first word of an external interrupt service routine is stored in location 3, and the first word of a timer/event counter interrupt service routine is stored in location 7.

Program memory can be used to store constants as well as program instructions. The MOVP instruction allows easy table lookup for constants and display formatting.

### 2.6.2 Data Memory

On-chip data memory is organized as 64 words eight bits wide. All locations are indirectly addressable and eight designated locations are directly addressable. Also included in the data memory is the program counter stack, addressed by a 3-bit stack pointer.

The first eight locations (0-7) of the array are designated as working registers and are directly addressable by any of the 11 direct register instructions. These locations are readily accessible for a variety of operations with a minimum number of instruction bytes required for their manipulation. Thus, they are usually used to store frequently accessed in-

termediate results. The DJNZ instruction makes very efficient use of the working registers as program loop counters by allowing the programmer to decrement and test the register in a single instruction.

Registers 0 and 1 have yet another function in that they can be used to indirectly address all locations in the data memory using the indirect register instructions. These two RAM pointer registers are especially useful for repetitive type operations on adjacent memory locations. The indirect register instruction specifies which register is used to address a location in RAM. The contents of the addressed location are used during the execution of the instruction and may be modified. The pointer registers may also point to registers 0-7, if desired.

Locations 8-23 serve a dual role in that they contain the 8-level program counter stack, two RAM locations per level. The program counter stack enables the processor to keep track of the return addresses generated by interrupts or CALL instructions by storing the contents of the program counter prior to servicing the subroutine. A 3-bit stack pointer determines which of the program counter stack's eight

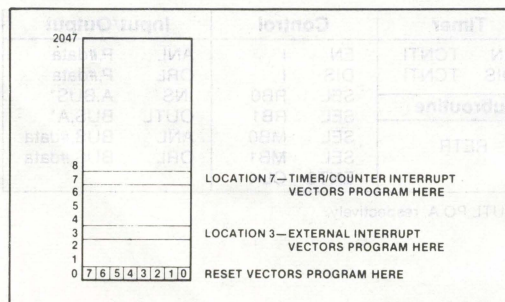


Figure 2-26. Program Memory Map

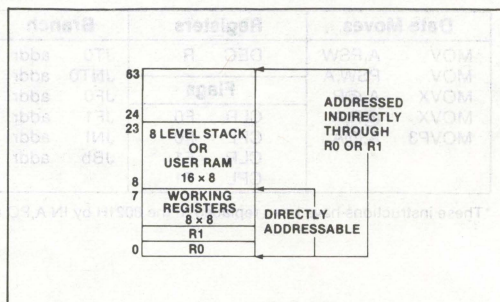


Figure 2-27. Data Memory Map



register pairs will be loaded with the next return address generated. The stack pointer, when initialized to 000 by RESET, points to RAM locations 8 and 9. The first subroutine CALL or interrupt results in the program counter contents being transferred to locations 8 and 9. The stack pointer is then incremented by one and points to locations 10 and 11 in anticipation of another CALL. The end of a subroutine, which is signaled by a return instruction (RET or RETI), causes the stack pointer to be decremented and the contents of the resulting register pair to be transferred to the program counter.

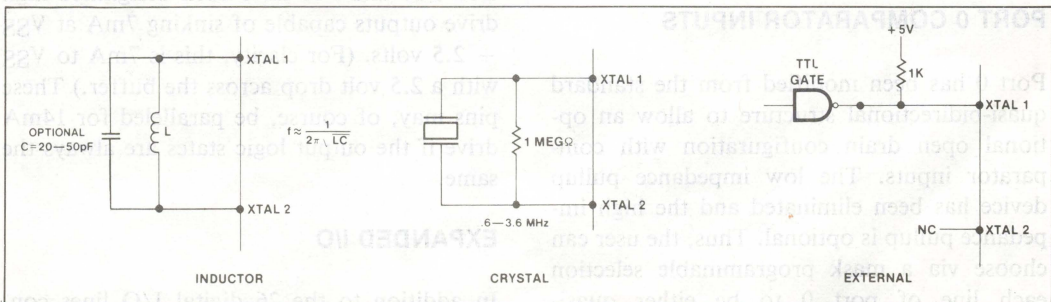
Unlike the 8048, in the 8022 the unincremented program counter address is stored in the address stack. The stack contents are then incremented before being loaded into the program counter during a return (RET) from subroutine. However, during a return (RETI) from interrupt, the stack contents are loaded directly into the program counter. This difference makes it imperative to use only RETI's to return from interrupts, and RET's to return from subroutines.

Since the program counter's addresses are 11 bits long, two bytes or registers must be used to store a single address. Thus, the 16-byte program counter stack permits up to a total of

8 levels of subroutine nesting without overflowing the stack. If overflow does occur, the deepest address stored (locations 8 and 9) will be overwritten and lost since the stack pointer overflows from 111 to 000. It also underflows from 000 to 111. If a particular application does not require 8 levels of nesting, the unused portion of the program counter stack may be used as any other indirectly addressable RAM location. For example, if only 3 levels of subroutine nesting are used, then only locations 8-13 need be reserved for the program counter stack, and locations 14-23 can be used for data storage.

## 2.6.3 Oscillator and Clock

The 8022 contains its own on-board oscillator and clock circuit, requiring only an external timing control element. See Figure 2-28. This control element can be a crystal, inductor, or clock. All internal time slots are derived from the external element, and all outputs are a function of the oscillator frequency. An instruction cycle consists of 10 states, and each state is a time slot of 3 oscillator periods. Therefore, to obtain a 10  $\mu$ s instruction cycle, a 3 MHz crystal should be used. The minimum instruction cycle time of 8.38  $\mu$ sec corresponds to a 3.58 MHz crystal.



**Figure 2-28. Frequency Reference Options**



### 2.6.4 Timer/Event Counter

Like the other MCS-48 microcomputers, the 8022 has an internal timer/event counter. This circuit can monitor elapsed time or count external events that occur during program execution. See the 8021H description, Section 2.5.4, for a complete explanation.

### 2.6.5 Input/Output Capabilities

The 8022 has 26 lines which can be used for digital input or output functions. These lines are organized as 3 ports of 8 lines, each of which serve as either inputs, outputs, or bidirectional ports, and 2 test inputs which can alter program sequences when tested by conditional jump instructions.

Ports 1 and 2 have identical operating characteristics and are both quasi-bidirectional. That is, each line may serve as an input, an output, or both. Data written to these ports is statically latched and remains unchanged until rewritten. As inputs, these lines are non-latching, i.e., inputs must be present until read by an input instruction. Inputs are fully TTL compatible and all outputs will drive at least one standard TTL load. See Section 2.1.4 for a more complete description of the quasi-bidirectional structure.

### PORT 0 COMPARATOR INPUTS

Port 0 has been modified from the standard quasi-bidirectional structure to allow an optional open drain configuration with comparator inputs. The low impedance pullup device has been eliminated and the high impedance pullup is optional. Thus, the user can choose via a mask programmable selection each line of port 0 to be either quasi-bidirectional with a high impedance or true

open drain. The open drain configuration allows the line to sink current through the low impedance pulldown device or to float in the high output state. More importantly, the open drain configuration makes port 0 very easy to drive when it is used as inputs. The input circuitry for each line of port 0 includes a voltage comparator which amplifies the voltage difference between the input port line and the port 0 threshold reference pin ( $V_{TH}$ ). The voltage gain of the comparator is sufficient to sense a 100 mV input differential within the range  $V_{SS}$  to  $V_{CC}/2$ .

If  $V_{TH}$  is allowed to float, it will bias itself to the digital switch point of the other ports, and port 0 behaves as a set of normal digital inputs. However, by biasing  $V_{TH}$ , the switch point can be both tightly controlled and adjusted. Common uses for this would include high noise margin inputs ( $V_{CC}/2$ ), unusual logic level inputs as from a diode isolated keyboard, analog channel expansion, and direct capacitive touchpanel interface. The comparator action is automatic and the port is read just as any other port.

### HIGH CURRENT OUTPUTS

High current drive is desirable for minimizing external parts required to do high power control. P10 and P11 have been designated high drive outputs capable of sinking 7mA at  $V_{SS} + 2.5$  volts. (For clarity, this is 7mA to  $V_{SS}$  with a 2.5 volt drop across the buffer.) These pins may, of course, be paralleled for 14mA drive if the output logic states are always the same.

### EXPANDED I/O

In addition to the 26 digital I/O lines contained on-board the 8022, a user can obtain



additional I/O lines by utilizing the Intel 8243 I/O expander chip or standard TTL. The 8243 interfaces to 4 port lines of the 8022 (lower half of port 2) and is strobed by the PROG line of the 8022.

The interface procedure is exactly the same as with the 8021H.

### 2.6.6 Test and Interrupt Inputs

In addition to the 24 general purpose I/O lines which comprise ports 0, 1, and 2, the 8022 has two inputs which are testable via conditional jump instructions, T0 and T1. These pins allow inputs to cause program branches without the necessity to load an input port into the accumulator. T0 and T1 have other functions as well.

The Test 0 pin serves as an external interrupt input as well as a testable input. An interrupt sequence is initiated by applying a low "0" level input to the T0 pin when external interrupt is enabled. Interrupt is level triggered and active low to allow "WIRE ORING" of several inter-

rupt sources at the input pin. When an interrupt is detected, it causes a "jump to subroutine" at location 3 in program memory as soon as all other cycles of the current instruction are complete. At this time, the program counter contents are saved in the program counter stack, but the remaining status of the processor is not. Unlike the 8048, the 8022 does not contain a program status word. Thus, when appropriate, the carry and auxiliary carry flags are saved in software, as is the accumulator. The routine shown below saves the accumulator and the carry flags in only four bytes.

| Instructions | Bytes | Comments                        |
|--------------|-------|---------------------------------|
| MOV R6,A     | 1     | ;save accumulator               |
| CLR A        | 1     | ;clear accumulator              |
| DA A         | 1     | ;convert carry flags into sixes |
| MOV R7,A     | 1     | ;save status of carry flags     |

The end of an interrupt service subroutine is marked by the execution of a Return from Interrupt instruction (RETI). Prior to returning from the interrupt subroutine, however, the status of the accumulator and the carry flags are restored in software. The following routine

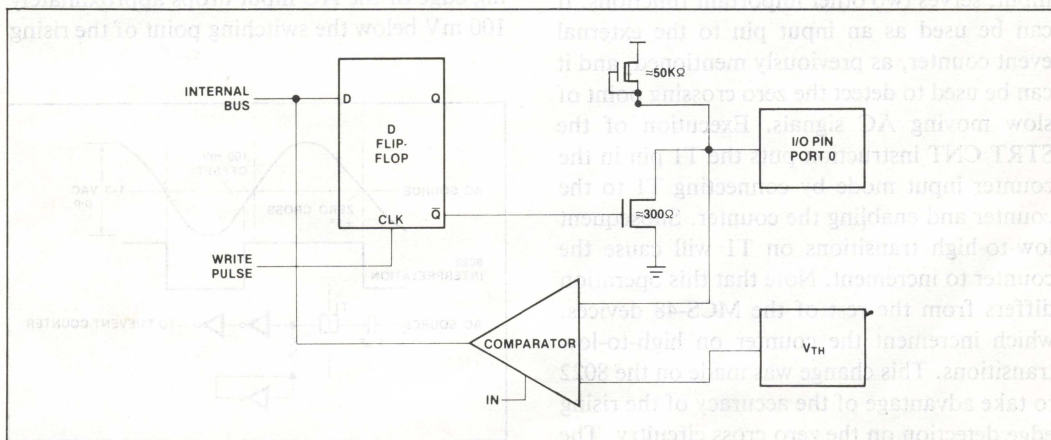


Figure 2-29. Port 0 I/O Structure



## SINGLE COMPONENT MCS-48 SYSTEM

restores the status of the accumulator and the carry flags, which was previously saved, in five bytes.

| Instructions | Bytes | Comments                               |
|--------------|-------|--|
| MOV A,R7     | 1     | ;restore carry flags status to         |
| Add A,#0AAH  | 2     | ;accumulator and set/clear carry flags |
| MOV A,R6     | 1     | ;restore accumulator                   |
| RETI         | 1     | ;return                                |

The interrupt system is single level in that once an interrupt is detected, all further interrupt requests are ignored until execution of a RETI re-enables the interrupt input logic. This sequence holds true also for an internal interrupt generated by timer overflow. If an external interrupt and an internal timer/counter generated interrupt are detected at the same time, the external source will be recognized. If needed, a second external interrupt can be created by enabling the timer/counter interrupt, loading FFH in the counter (one less than terminal count) and enabling the event counter mode. A low-to-high transition on the T1 input will then cause an interrupt vector to location 7.

The Test 1 pin, in addition to being a testable input, serves two other important functions. It can be used as an input pin to the external event counter, as previously mentioned, and it can be used to detect the zero crossing point of slow moving AC signals. Execution of the STRT CNT instruction puts the T1 pin in the counter input mode by connecting T1 to the counter and enabling the counter. Subsequent low-to-high transitions on T1 will cause the counter to increment. Note that this operation differs from the rest of the MCS-48 devices, which increment the counter on high-to-low transitions. This change was made on the 8022 to take advantage of the accuracy of the rising edge detection on the zero cross circuitry. The maximum rate at which the counter may be in-

cremented is once per three instruction cycles (every 30  $\mu$ s when using a 3 MHz crystal) – there is no minimum frequency.

In addition to serving as a testable input and as the counter input, the T1 pin has special circuitry to detect when an AC signal crosses its average DC level. When driven directly, this pin responds as a normal digital input. To utilize the zero cross detection mode, an AC signal of approximately 1-3 VAC p-p magnitude and a maximum frequency of 1 kHz is coupled through an external capacitor (1  $\mu$ F) to the T1 pin.

The internal digital state is sensed as a zero until the rising edge crosses the DC average level, when it becomes a one. This is accomplished by the self-biasing high gain amplifier which is included in the T1 input. This circuit biases the T1 input exactly at its switching point, such that a small change will cause a digital transition to occur. This digital transition takes place within 5 degrees of the zero point. The digital value of T1 remains a one until the falling edge of the AC input drops approximately 100 mV below the switching point of the rising

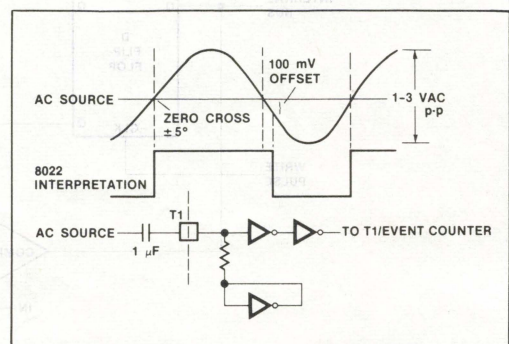


Figure 2-30. Zero-Cross Detection



## SINGLE COMPONENT MCS-48 SYSTEM

edge (100 mV below the zero point, if the digital transition occurred exactly at the zero point). The 100 mV offset is created by hysteresis and eliminates chattering of the internal signal caused by the external noise.

The zero cross detection capability allows the user to make the 60 Hz power signal the basis for this system timing. All timing routines, including time-of-day, can be implemented using the zero cross detection capability of T1 and its conditional jump instructions. In addition, the zero cross detection feature can be used in conjunction with the timer interrupt to interrupt processing at the zero voltage point. This enables the user to control voltage phase sensitive devices such as triacs and SCRs, and to use the 8022 in applications such as shaft speed and angle measurement.

### 2.6.7 Analog to Digital Converter

The 8022 contains on-chip a complete hardware implementation of an 8-bit analog to digital (A/D) converter with two multiplexed analog inputs. The A/D converter utilizes a successive approximation technique to provide an updated conversion once every four instruction cycles with a minimum of required software.

The A/D converter consists of four main parts, the input circuitry, a series string of resistors, a voltage comparator, and the successive approximation logic. The two analog inputs are multiplexed on-chip and selected via software by the SEL AN0 and SEL AN1 instructions. Besides selecting one of the analog inputs, these instructions restart the conversion sequence which operates continuously. Restarting a conversion sequence deletes the conversion in progress but does not affect the result of the previous conversion which is

stored in the conversion result register. The continuous operation of the A/D converter saves program space and time by allowing the user to obtain multiple readings from a given input with only one select instruction. To obtain a valid conversion reading, the user must provide the analog input signal no later than the beginning of the select instruction cycle. The analog input is then sampled by the A/D converter and maintained internally. This voltage becomes one input to the voltage comparator which amplifies the difference between the analog input and the voltage tap on the series resistor string.

The series resistor string is connected between the A/D reference pin ( $V_{AREF}$ ) and ground ( $AV_{SS}$ ). It is comprised of 256 identical resistors which divide the voltage between these two pins into 256 identical voltage steps. This configuration gives the converter its inherent monotonicity. The range of  $V_{AREF}$  in

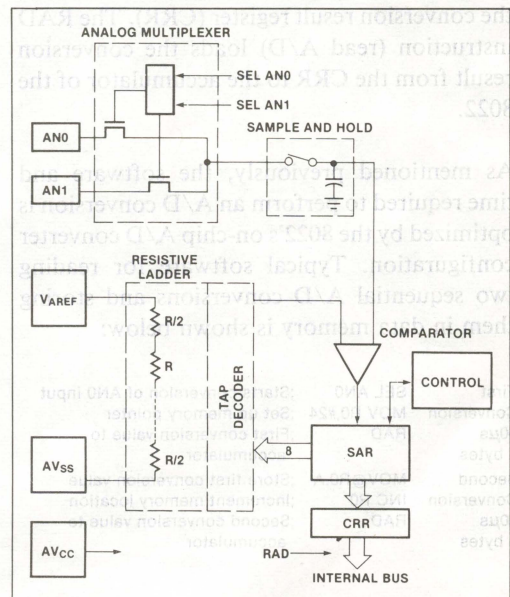


Figure 2-31. A/D Converter Block Diagram



## SINGLE COMPONENT MCS-48 SYSTEM

which full 8-bit resolution can be provided is between  $V_{CC}/2$  and  $V_{CC}$ .

Thus, the user is given a minimum voltage range from ground to  $V_{CC}/2$  and a maximum range from ground to  $V_{CC}$  over which 8-bit resolution is insured.

The voltage tap on the series resistor string is selected by the resistor ladder decoder. This decoder is driven by the 8-bit successive approximation register (SAR). Each bit of the SAR is set in succession, MSB to LSB, and a voltage comparison between the selected resistor ladder voltage and the analog input voltage is performed after the setting of each bit. The result of each comparison determines whether the particular bit will remain set or be reset. All comparisons are performed automatically by the on-chip A/D hardware. At the end of 8 comparisons the SAR contains a valid digital result which is then latched into the conversion result register (CRR). The RAD instruction (read A/D) loads the conversion result from the CRR to the accumulator of the 8022.

As mentioned previously, the software and time required to perform an A/D conversion is optimized by the 8022's on-chip A/D converter configuration. Typical software for reading two sequential A/D conversions and storing them in data memory is shown below:

|                   |            |   |
|-------------------|------------|---|
| First Conversion  | SEL AN0    | ;Starts conversion of AN0 input         |
| 50 $\mu$ s        | MOV R0,#24 | ;Set up memory pointer                  |
| 4 bytes           | RAD        | ;First conversion value to accumulator  |
| Second Conversion | MOV@R0,A   | ;Store first conversion value           |
| 40 $\mu$ s        | INC R0     | ;Increment memory location              |
| 3 bytes           | RAD        | ;Second conversion value to accumulator |

Note that the second conversion occurs without a second select instruction being used. Rather, the continuous operation of the A/D converter provides an updated digital value 4 instruction cycles after the first.

To insure maximum accuracy from the A/D converter, separate power supply pins (AVCC and AVSS) and a substrate pin (SUBST) have been provided. Supplying the power supply pins with a well filtered and regulated voltage supply minimizes the effect of power supply variance and system noise. The substrate pin should be bypassed to ground through a 500 pF to 0.001  $\mu$ F capacitor.

### 2.6.8 CPU

The 8022 CPU has arithmetic and logical capability. There is a wide variety of arithmetic and logic instructions which affect the contents of the accumulator, and/or direct or indirect scratchpad locations. Provisions have been made for simplified BCD arithmetic capability using the DAA, SWAP A, and XCHD instructions. In addition, MOVP A, @A allows table lookup for display formatting and constants. The conditional branch logic within the processor enables several conditions internal and external to the processor to be tested by the user's program. Use the conditional jump instructions with the tests listed below to effect a change in the program execution sequence.


| Test                | Jump Condition     | Jump Instructions |
|---------------------|--------------------|-------------------|
| Accumulator         | A = 0   A $\neq$ 0 | JZ JNZ            |
| Carry Flag          | 0   1              | JNC,JC            |
| Timer Overflow Flag | —   1              | JTF               |
| Test Input-T1       | 0   1              | JNT1, JT1         |
| Test Input-T0       | 0   1              | JNT0, JT0         |



## SINGLE COMPONENT MCS-48 SYSTEM

### 2.6.9 8022 Testing and Debugging

To facilitate testing and debug, certain test modes may be activated in the 8022 by raising combinations of RESET, TEST 1 and PROG to 15 volts. Internal ROM is dumped out sequentially for verification. External memory operation is used for CPU checkout.

| Reset | Prog | Test 1  | Case    | Function  |
|-------|------|---|---------|---|
| 5V    | X    | X   |         | Power On Clear  |
| 0V    | X    | X   |         | Normal Operation  |
| 15V   | 15V  | 15V   | Mode 1a | On each cycle internal ROM is dumped to Port 0 — Sequentially after ALE leading edge.             |
| 15V   | 15V  |  | Mode 1b | On every TEST 1 falling edge the program counter increments, dumps internal ROM to Port 0.        |
| 0V    | 15V  | X   | Mode 2  | Chip will operate from external memory (one page) via Port 0. ALE strobes Address out, memory in. |
| 15V   | X    | X   | Mode 3  | Chip accepts op codes into Port 1. Allows Port 0 and 8243 testing.                                |

#### NOTE

X = Normal mode - between 0V and  $V_{CC}$

Test 1 in Mode 1b should be limited to  $V_{CC}$



---

# Expanded MCS<sup>®</sup>-48 System 3

---



## CHAPTER 3

# EXPANDED MCS<sup>®</sup>-48 SYSTEM

### 3.0 INTRODUCTION

If the capabilities resident on the single-chip 8048H/8748/8035HL/8049H/8749H/8039HL are not sufficient for your system requirements, special on-board circuitry allows the addition of a wide variety of external memory, I/O, or special peripherals you may require. The processors can be directly and simply expanded in the following areas:

- Program Memory to 4K words
- Data Memory to 320 words (384 words with 8049H)
- I/O by unlimited amount
- Special Functions using 8080/8085AH peripherals

By using bank switching techniques, maximum capability is essentially unlimited. Bank switching is discussed later in the chapter. Expansion is accomplished in two ways:

- 1) Expander I/O—A special I/O Expander circuit, the 8243, provides for the addition of four 4-bit Input/Output ports with the sacrifice of only the lower half (4-bits) of port 2 for inter-device communication. Multiple 8243's may be added to this 4-bit bus by generating the required "chip select" lines.
- 2) Standard 8085 Bus—One port of the 8048H/8049H is like the 8-bit bidirectional data bus of the 8085 microcomputer system allowing interface to the numerous standard memories and peripherals of the MCS-80/85 microcomputer family.

MCS-48 systems can be configured using either or both of these expansion features to optimize system capabilities to the application.

Both expander devices and standard memories and peripherals can be added in virtually any number and combination required.

### 3.1 EXPANSION OF PROGRAM MEMORY

Program Memory is expanded beyond the resident 1K or 2K words by using the 8085 BUS feature of the MCS-48. All program memory fetches from the addresses less than 1024 on the 8048H and less than 2048 on the 8049H occur internally with no external signals being generated (except ALE which is always present). At address 1024 on the 8048H, the processor automatically initiates external program memory fetches.

#### 3.1.1 Instruction Fetch Cycle (External)

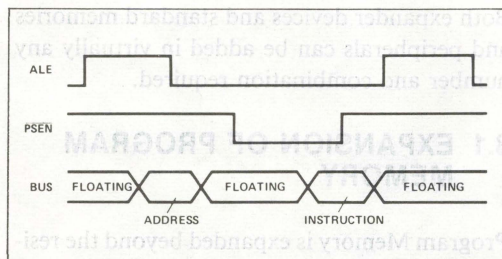
As shown in Figure 3-1, for all instruction fetches from addresses of 1024 (2048) or greater, the following will occur:

- 1) The contents of the 12-bit program counter will be output on BUS and the lower half of port 2.
- 2) Address Latch Enable (ALE) will indicate the time at which address is valid. The trailing edge of ALE is used to latch the address externally.
- 3) Program Store Enable ( $\overline{\text{PSEN}}$ ) indicates that an external instruction fetch is in progress and serves to enable the external memory device.
- 4) BUS reverts to input (floating) mode and the processor accepts its 8-bit contents as an instruction word.

All instruction fetches, including internal addresses, can be forced to be external by ac-



## EXPANDED MCS-48 SYSTEM



**Figure 3-1. Instruction Fetch from External Program Memory**

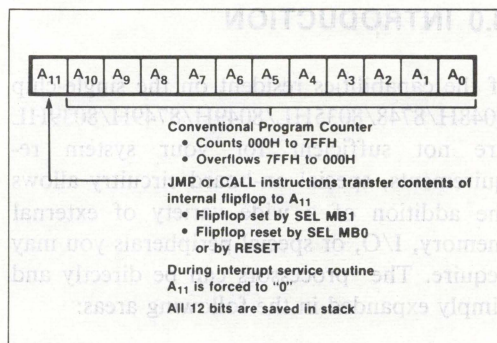
tivating the EA pin of the 8048H/8049H. The 8035HL/8039HL processors without program memory always operate in the external program memory mode (EA = 5V).

### 3.1.2 Extended Program Memory Addressing (Beyond 2K)

For programs of 2K words or less, the 8048H/8049H addresses program memory in the conventional manner. Addresses beyond 2047 can be reached by executing a program memory bank switch instruction (SEL MB0, SEL MB1) followed by a branch instruction (JMP or CALL). The bank switch feature extends the range of branch instructions beyond their normal 2K range and at the same time prevents the user from inadvertently crossing the 2K boundary.

#### PROGRAM MEMORY BANK SWITCH

The switching of 2K program memory banks is accomplished by directly setting or resetting the most significant bit of the program counter (bit 11); see Figure 3-2. Bit 11 is not altered by normal incrementing of the program counter but is loaded with the contents of a special flip-flop each time a JMP or CALL instruction is executed. This special flip-flop is set by ex-



**Figure 3-2. Program Counter**

ecuting an SEL MB1 instruction and reset by SEL MB0. Therefore, the SEL MB instruction may be executed at any time prior to the actual bank switch which occurs during the next branch instruction encountered. Since all twelve bits of the program counter, including bit 11, are stored in the stack, when a Call is executed, the user may jump to subroutines across the 2K boundary and the proper bank will be restored upon return. However, the bank switch flip-flop will not be altered on return.

#### INTERRUPT ROUTINES

Interrupts always vector the program counter to location 3 or 7 in the first 2K bank, and bit 11 of the program counter is held at "0" during the interrupt service routine. The end of the service routine is signalled by the execution of an RETR instruction. Interrupt service routines should therefore be contained entirely in the lower 2K words of program memory. The execution of a SEL MB0 or SEL MB1 instruction within an interrupt routine is not recommended since it will not alter PC11 while in the routine, but will change the internal flip-flop.



## EXPANDED MCS-48 SYSTEM

### 3.1.3 Restoring I/O Port Information

Although the lower half of Port 2 is used to output the four most significant bits of address during an external program memory fetch, the I/O information is still outputted during certain portions of each machine cycle. I/O information is always present on Port 2's lower 4 bits at the rising edge of ALE and can be sampled or latched at this time.

### 3.1.4 Expansion Examples

Shown in Figure 3-3 is the addition of 2K words of program memory using an 2716A 2K x 8 ROM to give a total of 3K words of program memory. In this case no chip select decoding is required and  $\overline{\text{PSEN}}$  enables the memory directly through the chip select input. If the system requires only 2K of program memory, the same configuration can be used with an 8035HL substituted for the 8048H. The 8049H would provide 4K of program memory with the same configuration.

Figure 3-4 shows how the 8755/8355 EPROM/ROM with I/O interfaces directly to the 8048H without the need for an address latch. The 8755/8355 contains an internal 8-bit address latch eliminating the need for an 8212 latch. In addition to a 2K x 8 program memory, the 8755/8355 also contains 16 I/O lines addressable as two 8-bit ports. These ports are addressed as external RAM; therefore the  $\overline{\text{RD}}$  and  $\overline{\text{WR}}$  outputs of the 8048H are required. See the following section on data memory expansion for more detail. The subsequent section on I/O expansion explains the operation of the 16 I/O lines.

## 3.2 EXPANSION OF DATA MEMORY

Data Memory is expanded beyond the resident 64 words by using the 8085AH type bus feature of the MCS-48.

### 3.2.1 Read/Write Cycle

All address and data is transferred over the 8

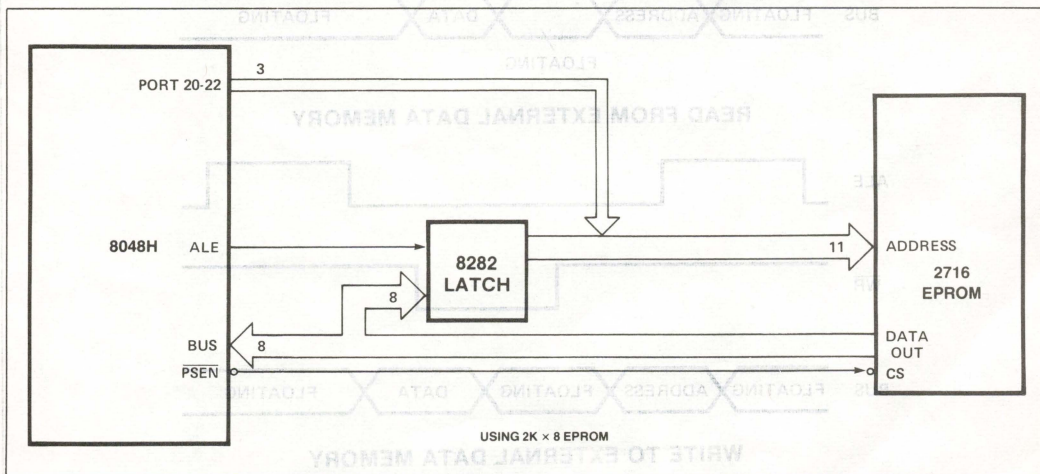
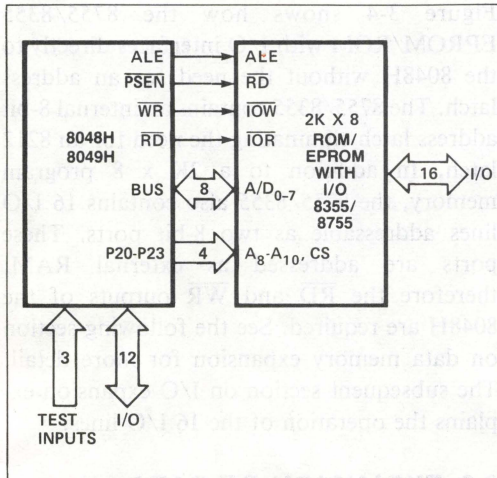


Figure 3-3. Expanding MCS-48 Program Memory Using Standard Memory Products



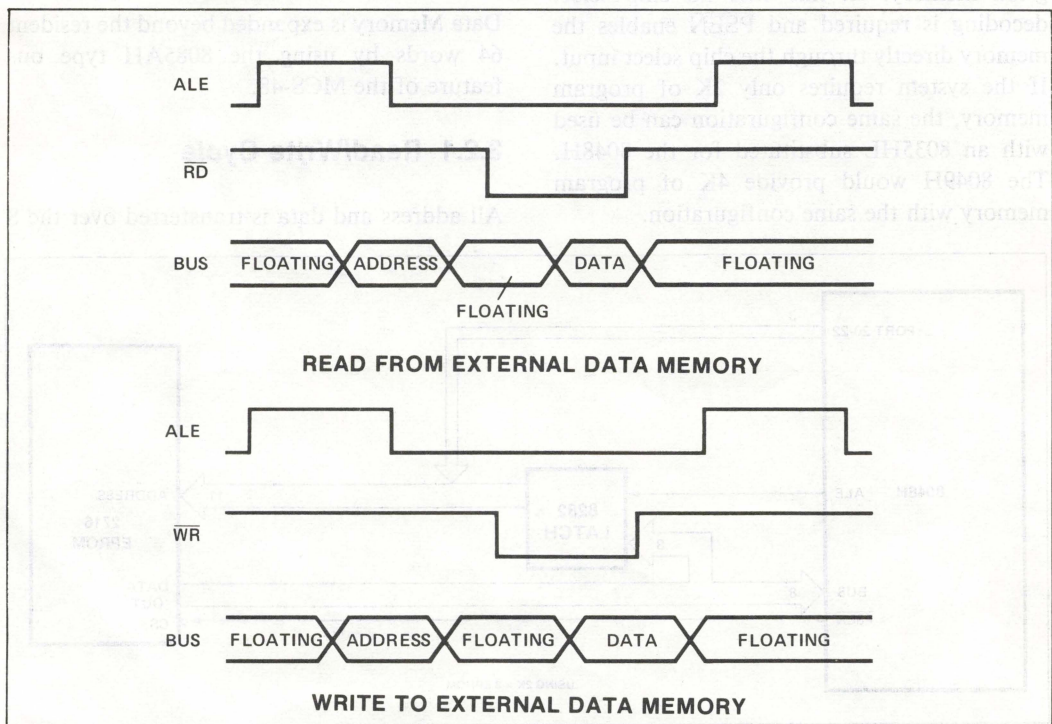
## EXPANDED MCS-48 SYSTEM



**Figure 3-4. External Program Memory Interface**

lines of BUS. As shown in Figure 3-5, a read or write cycle occurs as follows:

- 1) The contents of register R0 or R1 is outputted on BUS.
- 2) Address Latch Enable (ALE) indicates address is valid. The trailing edge of ALE is used to latch the address externally.
- 3) A read ( $\overline{RD}$ ) or write ( $\overline{WR}$ ) pulse on the corresponding output pins of the 8048H indicates the type of data memory access in progress. Output data is valid at the trailing edge of  $\overline{WR}$  and input data must be valid at the trailing edge of  $\overline{RD}$ .
- 4) Data (8 bits) is transferred in or out over BUS.



**Figure 3-5. External Data Memory Timings**



## EXPANDED MCS-48 SYSTEM

### 3.2.2 Addressing External Data Memory

External Data Memory is accessed with its own two-cycle move instructions, MOVX A, @R and MOVX @R, A, which transfer 8 bits of data between the accumulator and the external memory location addressed by the contents of one of the RAM Pointer Registers R0 and R1. This allows 256 locations to be addressed in addition to the resident locations. Additional pages may be added by "bank switching" with extra output lines of the 8048H.

### 3.2.3 Examples of Data Memory Expansion

Figure 3-6 shows how the 8048H can be expanded using the 8155 memory and I/O expanding device. Since the 8155 has an internal 8-bit address latch, it can interface directly to the 8048H without the use of an external latch. The 8155 provides an additional 256 words of static data memory and also includes 22 I/O lines and a 14-bit timer. See the following section on I/O expansion and the 8155 data sheet for more details on these additional features.

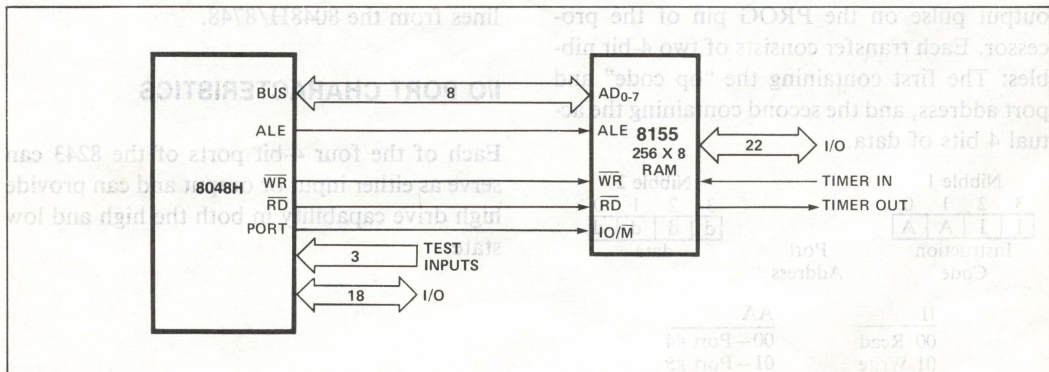


Figure 3-6. 8048H Interface to 256 × 8 Standard Memories

### 3.3 EXPANSION OF INPUT/OUTPUT

There are four possible modes of I/O expansion with the 8048H: one using a special low-cost expander, the 8243; another using standard MCS-80/85 I/O devices; and a third using the combination memory/I/O expander devices the 8155, 8355, and 8755. It is also possible to expand using standard TTL devices as shown in Chapter 5.

#### 3.3.1 I/O Expander Device

The most efficient means of I/O expansion for small systems is the 8243 I/O Expander Device which requires only 4 port lines (lower half of Port 2) for communication with the 8048H. The 8243 contains four 4-bit I/O ports which serve as an extension of the on-chip I/O and are addressed as ports #4-7 (see Figure 3-7). The following operations may be performed on these ports:

- Transfer Accumulator to Port
- Transfer Port to Accumulator
- AND Accumulator to Port
- OR Accumulator to Port



A high to low transition of the PROG line indicates that address is present, while a low to high transition indicates the presence of data. Additional 8243's may be added to the four-bit bus and chip selected using additional output lines from the 8048H/8748.

Nibble 1

|   |   |   |   |
|---|---|---|---|
| 3 | 2 | 1 | 0 |
| 1 | 1 | A | A |

Instruction  
Code

Nibble 2

|   |   |   |   |
|---|---|---|---|
| 3 | 2 | 1 | 0 |
| d | d | d | d |

Port  
Address

data

| <u>II</u> | <u>AA</u>  |
|-----------|------------|
| 00 Read   | 00—Port #4 |
| 01 Write  | 01—Port #5 |
| 10 OR     | 10—Port #6 |
| 11 AND    | 11—Port #7 |



## EXPANDED MCS-48 SYSTEM

### 3.3.2 I/O Expansion with Standard Peripherals

Standard MCS-80/85 type I/O devices may be added to the MCS-48 using the same bus and timing used for Data Memory expansion. Figure 3-8 shows an example of how an 8048H can be connected to an MCS-85 peripheral. I/O devices reside on the Data Memory bus and in the data memory address space and are accessed with the same MOVX instructions. (See the previous section on data memory expansion for a description of timing.) The following are a few of the Standard MCS-80 devices which are very useful in MCS-48 systems:

- 8214 Priority Interrupt Encoder
- 8251 Serial Communications Interface
- 8255 General Purpose Programmable I/O
- 8279 Keyboard/Display Interface
- 8253 Interval Timer

### 3.3.3 Combination Memory and I/O Expanders

As mentioned in the sections on program and data memory expansion, the 8355/8755 and 8155 expanders also contain I/O capability.

**8355/8755:** These two parts of ROM and EPROM equivalents and therefore contain the same I/O structure. I/O consists of two 8-bit ports which normally reside in the external data memory address space and are accessed with MOVX instructions. Associated with each port is an 8-bit Data Direction Register which defines each bit in the port as either an input or an output. The data direction registers are directly addressable, thereby allowing the user to define under software control each individual bit of the ports as either input or output. All outputs are statically latched and double buffered. Inputs are not latched.

**8155/8156:** I/O on the 8155/8156 is configured as two 8-bit programmable I/O ports and one 6-bit programmable port. These three

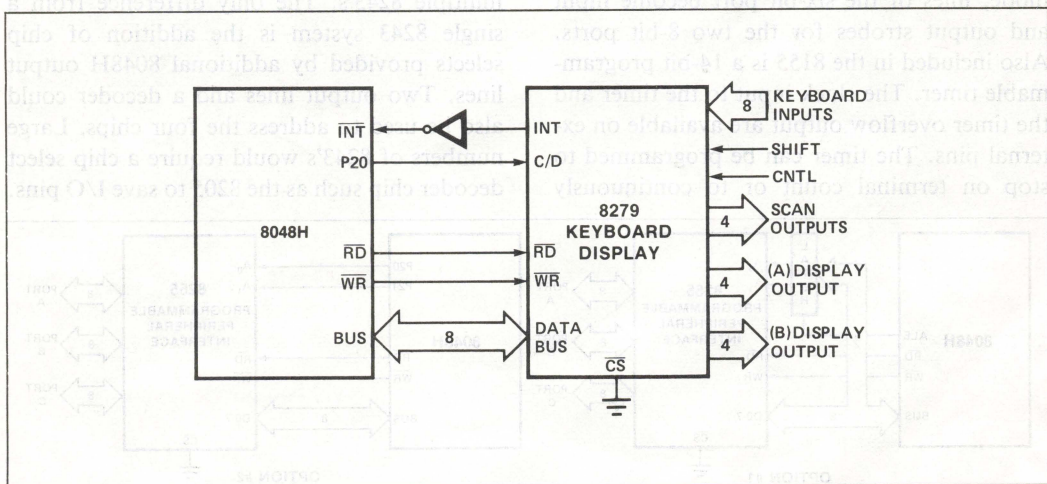
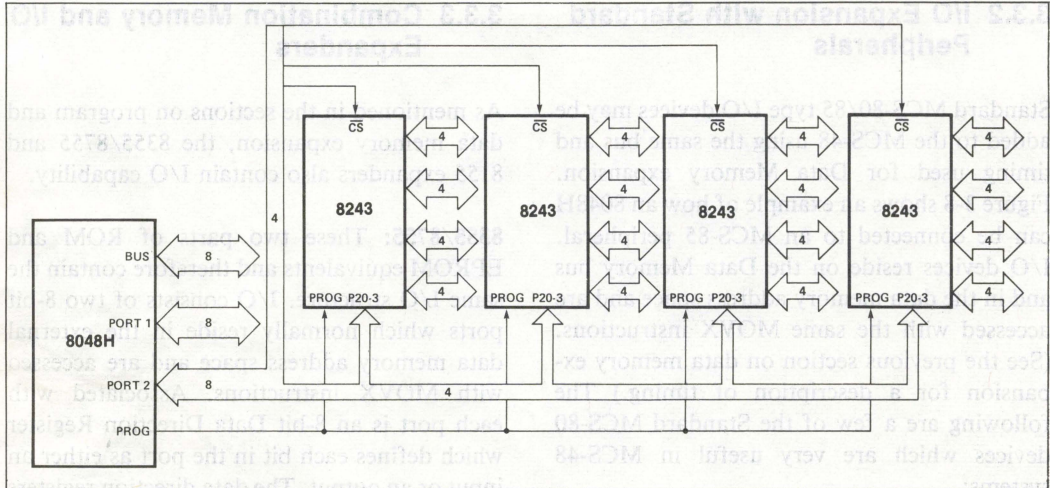


Figure 3-8. Keyboard/Display Interface



## EXPANDED MCS-48 SYSTEM



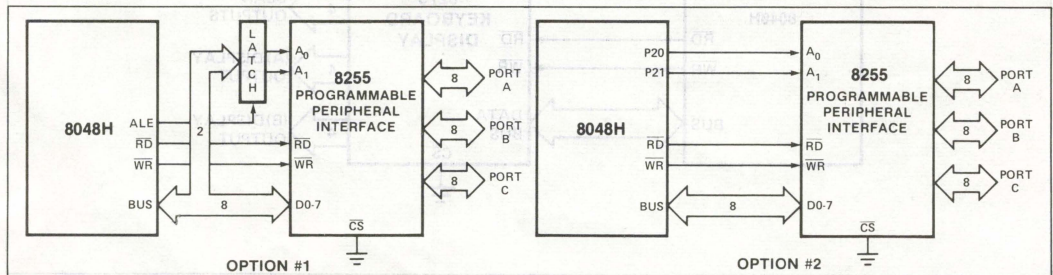
**Figure 3-9. Low Cost I/O Expansion**

registers and a Control/Status register are accessible as external data memory with the MOVX instructions. The contents of the control register determines the mode of the three ports. The ports can be programmed as input or output with or without associated handshake communication lines. In the handshake mode, lines of the six-bit port become input and output strobes for the two 8-bit ports. Also included in the 8155 is a 14-bit programmable timer. The clock input to the timer and the timer overflow output are available on external pins. The timer can be programmed to stop on terminal count or to continuously

reload itself. A square wave or pulse output on terminal count can also be specified.

### I/O EXPANSION EXAMPLES (SEE ALSO CHAPTER 5)

Figure 3-9 shows the expansion of I/O using multiple 8243's. The only difference from a single 8243 system is the addition of chip selects provided by additional 8048H output lines. Two output lines and a decoder could also be used to address the four chips. Large numbers of 8243's would require a chip select decoder chip such as the 8205 to save I/O pins.



**Figure 3-10. Interface to MCS-80 Peripherals**



## EXPANDED MCS-48 SYSTEM

Figure 3-10 shows the 8048H interface to a standard MCS-80 peripheral; in this case, the 8255 Programmable Peripheral Interface, a 40-pin part which provides three 8-bit programmable I/O ports. The 8255 bus interface is typical of programmable MCS-80 peripherals with an 8-bit bidirectional data bus, a  $\overline{RD}$  and  $\overline{WR}$  input for Read/Write control, a  $\overline{CS}$  (chip select) input used to enable the Read/Write control logic and the address inputs used to select various internal registers.

Interconnection to the 8048H is very straightforward with BUS,  $\overline{RD}$ , and  $\overline{WR}$  connecting directly to the corresponding pins on the 8255. The only design consideration is the way in which the internal registers of the 8255 are to be addressed. If the registers are to be

addressed as external data memory using the MOVX instructions, the appropriate number of address bits (in this case, 2) must be latched on BUS using ALE as described in the section on external data memories. If only a single device is connected to BUS, the 8255 may be continuously selected by grounding  $\overline{CS}$ . If multiple 8255's are used, additional address bits can be latched and used as chip selects.

A second addressing method eliminates external latches and chip select decoders by using output port lines as address and chip select lines directly. This method, of course, requires the setting of an output port with address information prior to executing a MOVX instruction.

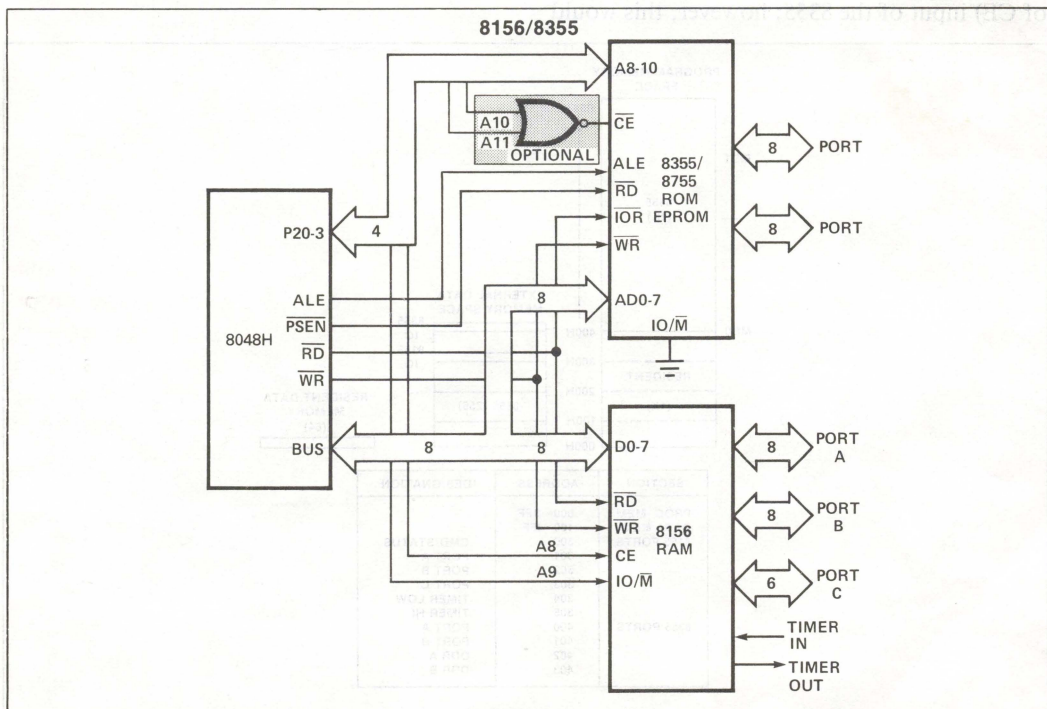


Figure 3-11. The Three-Component MCS-48 System



## EXPANDED MCS-48 SYSTEM

### 3.4 MULTI-CHIP MCS-48 SYSTEMS

Figure 3-11 shows the addition of two memory expanders to the 8048H, one 8355/8755 ROM and one 8156 RAM. The main consideration in designing such a system is the addressing of the various memories and I/O ports. Note that in this configuration address lines  $A_{10}$  and  $A_{11}$  have been ORed to chip select the 8355. This ensures that the chip is active for all external program memory fetches in the 1K to 3K range and is disabled for all other addresses. This gating has been added to allow the I/O port of the 8355 to be used. If the chip was left selected all the time, there would be conflict between these ports and the RAM and I/O of the 8156. The NOR gate could be eliminated and  $\overline{A}_{11}$  connected directly to the CE (instead of  $\overline{CE}$ ) input of the 8355; however, this would

create a 1K word "hole" in the program memory by causing the 8355 to be active in the 2K and 4K range instead of the normal 1K to 3K range.

In this system the various locations are addressed as follows:

- Data RAM—Addresses 0 to 255 when Port 2 Bit 0 has been previously set = 1 and Bit 1 set = 0
- RAM I/O—Addresses 0 to 3 when Port 2 Bit 0 = 1 and Bit 1 = 1
- ROM I/O—Addresses 0 to 3 when Port 2 Bit 2 or Bit 3 = 1

See the memory map in Figure 3-12.

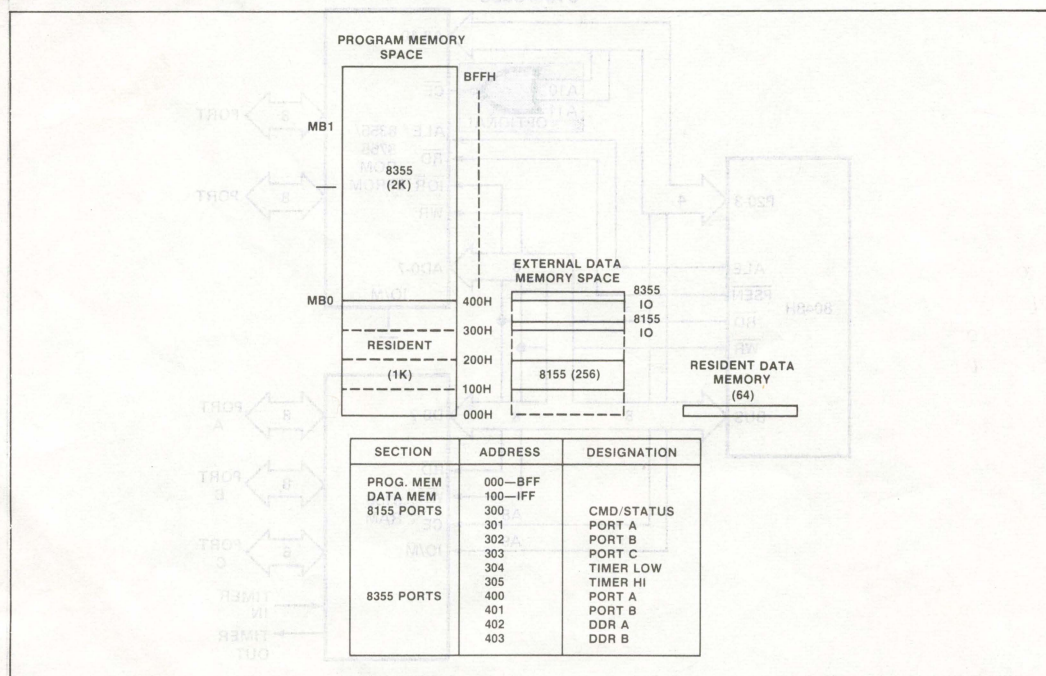
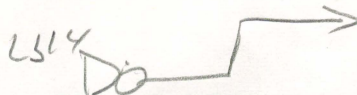


Figure 3-12. Memory Map for Three-Component MCS-48 Family





## EXPANDED MCS-48 SYSTEM

### 3.5 MEMORY BANK SWITCHING

Certain systems may require more than the 4K words of program memory which are directly addressable by the program counter or more than the 256 data memory and I/O locations directly addressable by the pointer registers R0 and R1. These systems can be achieved using "bank switching" techniques. Bank switching is merely the selection of various blocks or "banks" of memory using dedicated output port lines from the processor. In the case of the 8048H, program memory is selected in blocks of 4K words at a time, while data memory and I/O are enabled 256 words at a time.

The most important consideration in implementing two or more banks is the software required to cross the bank boundaries. Each crossing of the boundary requires that the processor first write a control bit to an output port before accessing memory or I/O in the new bank. If program memory is being switched, programs should be organized to keep boundary crossings to a minimum. Jumping to subroutines across the boundary should be avoided when possible since the programmer must keep track of which bank to return to after completion of the subroutine. If these subroutines are to be nested and accessed from either bank, a software "stack" should be implemented to save the bank switch bit just as if it were another bit of the program counter.

From a hardware standpoint bank switching is very straightforward and involves only the connection of an I/O line or lines as bank enable signals. These enables are ANDed with normal memory and I/O chip select signals to activate the proper bank.

### 3.6 CONTROL SIGNAL SUMMARY

Table 3-1 summarizes the instructions which activate the various control outputs of the MCS-48 processors. During all other instructions these outputs are driven to the inactive state.

Table 3-1. MCS-48 Control Signals

| Control Signal | When Active   |
|----------------|---|
| RD             | During MOVX A,@R or INS Bus   |
| WR             | During MOVX @R, A or OUTL Bus   |
| ALE            | Every Machine Cycle   |
| PSEN           | During Fetch of external program memory (instruction or immediate data) |
| PROG           | During MOVD A,P ANLD P,A<br>MOVD P,A ORLD P,A                           |

### 3.7 PORT CHARACTERISTICS

#### 3.7.1 BUS Port Operations

The BUS port can operate in three different modes: as a latched I/O port, as a bidirectional bus port, or as a program memory address output when external memory is used. The BUS port lines are either active high, active low, or high impedance (floating).

The latched mode (INS, OUTL) is intended for use in the single-chip configuration where BUS is not being used as an expander port. OUTL and MOVX instructions can be mixed if necessary. However, a previously latched output will be destroyed by executing a MOVX instruction and BUS will be left in the high impedance state. INS does not put the BUS in a high impedance state. Therefore, the use of MOVX after OUTL to put the BUS in a high impedance state is necessary before an INS instruction intended to read an external word (as opposed to the previously latched value).

3373



## EXPANDED MCS-48 SYSTEM

OUTL should never be used in a system with external program memory, since latching BUS can cause the next instruction, if external, to be fetched improperly.

### 3.7.2 Port 2 Operations

The lower half of Port 2 can be used in three different ways: as a quasi-bidirectional static port, as an 8243 expander port, and to address external program memory. In all cases outputs are driven low by an active device and driven high momentarily by a low impedance device and held high by a high impedance device to VCC.

The port may contain latched I/O data prior to its use in another mode without affecting operation of either. If lower Port 2 (P20-3) is used to output address for an external program memory fetch, the I/O information previously latched will be automatically removed temporarily while address is present, then restored when the fetch is complete. However, if lower Port 2 is used to communicate with an 8243, previously latched I/O information will be removed and not restored. After an input from the 8243, P20-3 will be left in the input mode (floating). After an output to the 8243, P20-3 will contain the value written, ANDed, or ORed to the 8243 port.

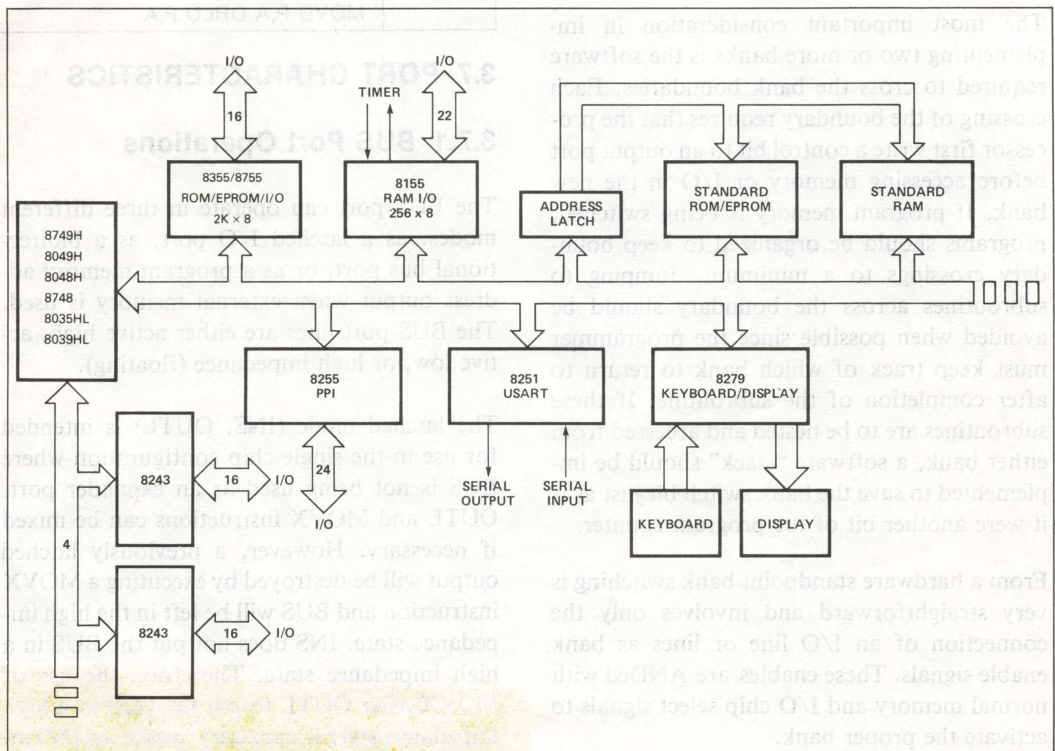


Figure 3-13. MCS-48 Expansion Capability



---

# MCS<sup>®</sup>-48 Instruction Set

---

4



# CHAPTER 4

## MCS-48<sup>®</sup> INSTRUCTION SET

### 4.0 INTRODUCTION

The MCS-48 instruction set is extensive for a machine of its size and has been tailored to be straightforward and very efficient in its use of program memory. All instructions are either one or two bytes in length and over 70% are only one byte long. Also, all instructions execute in either one or two cycles and over 50% of all instructions execute in a single cycle. Double cycle instructions include all immediate instructions, and all I/O instructions.

The MCS-48 microcomputers have been designed to handle arithmetic operations efficiently in both binary and BCD as well as han-

dle the single-bit operations required in control applications. Special instructions have also been included to simplify loop counters, table lookup routines, and N-way branch routines.

### 4.0.1 Data Transfers

As can be seen in Figure 4-1, the 8-bit accumulator is the central point for all data transfers within the 8048. Data can be transferred between the 8 registers of each working register bank and the accumulator directly, i.e., the source or destination register is specified by the instruction. The remaining locations of the internal RAM array are referred to as Data Memory and are addressed

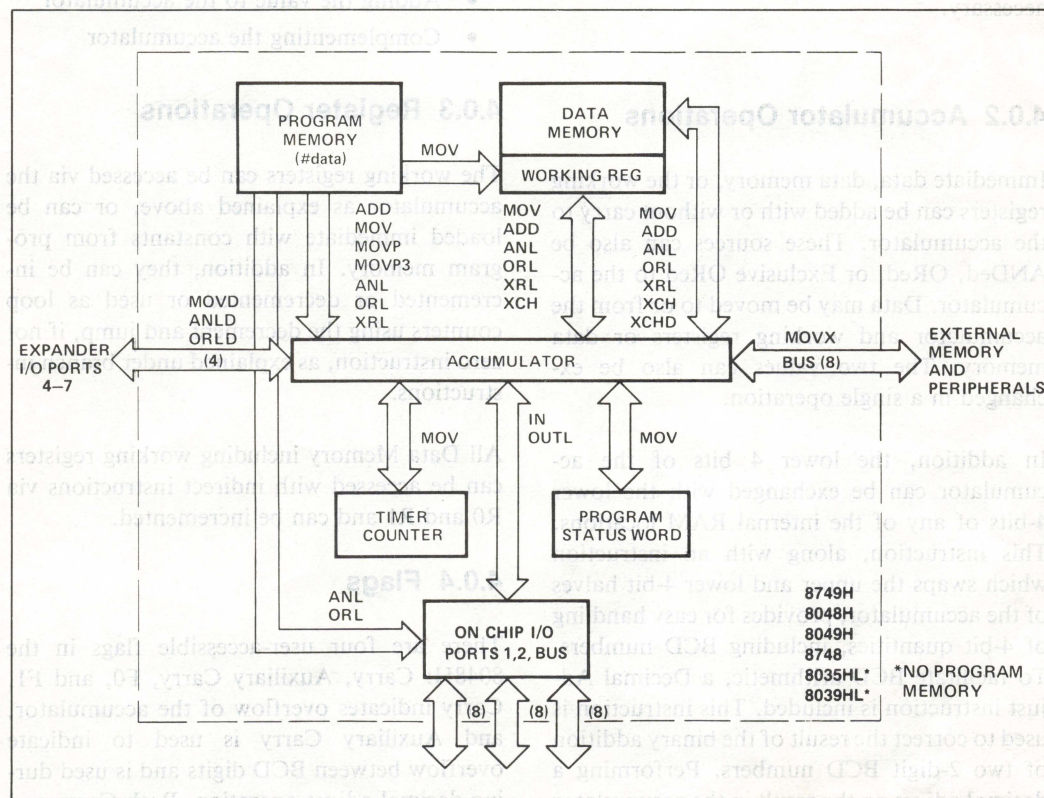


Figure 4-1. Data Transfer Instructions

AFN-02174A



## MCS-48 INSTRUCTION SET

indirectly via an address stored in either R0 or R1 of the active register bank. R0 and R1 are also used to indirectly address external data memory when it is present. Transfers to and from internal RAM require one cycle, while transfers to external RAM require two. Constants stored in Program Memory can be loaded directly to the accumulator and to the 8 working registers. Data can also be transferred directly between the accumulator and the on-board timer/counter or the accumulator and the Program Status word (PSW). Writing to the PSW alters machine status accordingly and provides a means of restoring status after an interrupt or of altering the stack pointer if necessary.

### 4.0.2 Accumulator Operations

Immediate data, data memory, or the working registers can be added with or without carry to the accumulator. These sources can also be ANDed, ORed, or Exclusive ORed to the accumulator. Data may be moved to or from the accumulator and working registers or data memory. The two values can also be exchanged in a single operation.

In addition, the lower 4 bits of the accumulator can be exchanged with the lower 4-bits of any of the internal RAM locations. This instruction, along with an instruction which swaps the upper and lower 4-bit halves of the accumulator, provides for easy handling of 4-bit quantities, including BCD numbers. To facilitate BCD arithmetic, a Decimal Adjust instruction is included. This instruction is used to correct the result of the binary addition of two 2-digit BCD numbers. Performing a decimal adjust on the result in the accumulator produces the required BCD result.

Finally, the accumulator can be incremented, decremented, cleared, or complemented and can be rotated left or right 1 bit at a time with or without carry.

Although there is no subtract instruction in the 8048H, this operation can be easily implemented with three single-byte single-cycle instructions.

A value may be subtracted from the accumulator with the result in the accumulator by:

- Complementing the accumulator
- Adding the value to the accumulator
- Complementing the accumulator

### 4.0.3 Register Operations

The working registers can be accessed via the accumulator as explained above, or can be loaded immediate with constants from program memory. In addition, they can be incremented or decremented or used as loop counters using the decrement and jump, if not zero instruction, as explained under branch instructions.

All Data Memory including working registers can be accessed with indirect instructions via R0 and R1 and can be incremented.

### 4.0.4 Flags

There are four user-accessible flags in the 8048H: Carry, Auxiliary Carry, F0, and F1. Carry indicates overflow of the accumulator, and Auxiliary Carry is used to indicate overflow between BCD digits and is used during decimal-adjust operation. Both Carry and Auxiliary Carry are accessible as part of the



## MCS-48 INSTRUCTION SET

---

program status word and are stored on the stack during subroutines. F0 and F1 are undedicated general-purpose flags to be used as the programmer desires. Both flags can be cleared or complemented and tested by conditional jump instructions. F0 is also accessible via the Program Status word and is stored on the stack with the carry flags.

### 4.0.5 Branch Instructions

The unconditional jump instruction is two bytes and allows jumps anywhere in the first 2K words of program memory. Jumps to the second 2K of memory (4K words are directly addressable) are made first by executing a select memory bank instruction, then executing the jump instruction. The 2K boundary can only be crossed via a jump or subroutine call instruction, i.e., the bank switch does not occur until a jump is executed. Once a memory bank has been selected all subsequent jumps will be to the selected bank until another select memory bank instruction is executed. A subroutine in the opposite bank can be accessed by a select memory bank instruction followed by a call instruction. Upon completion of the subroutine, execution will automatically return to the original bank; however, unless the original bank is reselected, the next jump instruction encountered will again transfer execution to the opposite bank.

Conditional jumps can test the following inputs and machine status:

- T0 Input pin
- T1 Input pin
- $\overline{\text{INT}}$  Input Pin
- Accumulator Zero
- Any bit of Accumulator

- Carry Flag
- F0 Flag
- F1 Flag

Conditional jumps allow a branch to any address within the current page (256 words) of execution. The conditions tested are the instantaneous values at the time the conditional jump is executed. For instance, the jump on accumulator zero instruction tests the accumulator itself, not an intermediate zero flag.

The decrement register and jump if not zero instruction combines a decrement and a branch instruction to create an instruction very useful in implementing a loop counter. This instruction can designate any one of the 8 working registers as a counter and can effect a branch to any address within the current page of execution.

A single-byte indirect jump instruction allows the program to be vectored to any one of several different locations based on the contents of the accumulator. The contents of the accumulator points to a location in program memory which contains the jump address. The 8-bit jump address refers to the current page of execution. This instruction could be used, for instance, to vector to any one of several routines based on an ASCII character which has been loaded in the accumulator. In this way ASCII key inputs can be used to initiate various routines.

### 4.0.6 Subroutines

Subroutines are entered by executing a call instruction. Calls can be made like unconditional jumps to any address in a 2K word bank, and jumps across the 2K boundary are executed in the same manner. Two separate



## MCS-48 INSTRUCTION SET

---

return instructions determine whether or not status (upper 4-bits of PSW) is restored upon return from the subroutine.

The return and restore status instruction also signals the end of an interrupt service routine if one has been in progress.

### 4.0.7 Timer Instructions

The 8-bit on board timer/counter can be loaded or read via the accumulator while the counter is stopped or while counting. The counter can be started as a timer with an internal clock source or as an event counter or timer with an external clock applied to the T1 input pin. The instruction executed determines which clock source is used. A single instruction stops the counter whether it is operating with an internal or an external clock source. In addition, two instructions allow the timer interrupt to be enabled or disabled.

### 4.0.8 Control Instructions

Two instructions allow the external interrupt source to be enabled or disabled. Interrupts are initially disabled and are automatically disabled while an interrupt service routine is in progress and re-enabled afterward.

There are four memory bank select instructions, two to designate the active working register bank and two to control program memory banks. The operation of the program memory bank switch is explained in section 3.1.2. The working register bank switch instructions allow the programmer to immediately substitute a second 8-register working register bank for the one in use. This effectively provides 16 working registers or it can be used as a means of quickly saving the contents

of the registers in response to an interrupt. The user has the option to switch or not to switch banks on interrupt. However, if the banks are switched, the original bank will be automatically restored upon execution of a return and restore status instruction at the end of the interrupt service routine.

A special instruction enables an internal clock, which is the XTAL frequency divided by three to be output on pin T0. This clock can be used as a general-purpose clock in the user's system. This instruction should be used only to initialize the system since the clock output can be disabled only by application of system reset.

### 4.0.9 Input/Output Instructions

Ports 1 and 2 are 8-bit static I/O ports which can be loaded to and from the accumulator. Outputs are statically latched but inputs are not latched and must be read while inputs are present. In addition, immediate data from program memory can be ANDed or ORed directly to Port 1 and Port 2 with the result remaining on the port. This allows "masks" stored in program memory to selectively set or reset individual bits of the I/O ports. Ports 1 and 2 are configured to allow input on a given pin by first writing a "1" out to the pin.

An 8-bit port called BUS can also be accessed via the accumulator and can have statically latched outputs as well. It too can have immediate data ANDed or ORed directly to its outputs, however, unlike ports 1 and 2, all eight lines of BUS must be treated as either input or output at any one time. In addition to being a static port, BUS can be used as a true synchronous bi-directional port using the Move External instructions used to access external data memory. When these instructions



## MCS-48 INSTRUCTION SET

are executed, a corresponding **READ** or **WRITE** pulse is generated and data is valid only at that time. When data is not being transferred, BUS is in a high impedance state.

The basic three on-board I/O ports can be expanded via a 4-bit expander bus using half of port 2. I/O expander devices on this bus consist of four 4-bit ports which are addressed as ports 4 through 7. These ports have their own **AND** and **OR** instructions like the on-board ports as well as move instructions to transfer data in or out. The expander **AND** and **OR** instructions, however, combine the contents of accumulator with the selected port rather than immediate data as is done with the on-board ports.

I/O devices can also be added externally using the BUS port as the expansion bus. In this case the I/O ports become "memory mapped", i.e., they are addressed in the same way as external data memory and exist in the external data memory address space addressed by pointer register R0 or R1.

### 4.1 INSTRUCTION SET DESCRIPTION

The following pages describe the MCS-48 instruction set in detail. The instruction set is first summarized with instructions grouped functionally. This summary page is followed by a detailed description listed alphabetically by mnemonic opcode.

The alphabetical listing includes the following information:

- Mnemonic
- Machine Code
- Verbal Description

- Symbolic Description
- Assembly Language Example

The machine code is represented with the most significant bit (7) to the left and two byte instructions are represented with the first byte on the left. The assembly language examples are formulated as follows:

Arbitrary  
Label: Mnemonic, Operand;  
Descriptive Comment

| Machine Code | Symbolic Description | Assembly Language Example |
|--------------|----------------------|---------------------------|
| 00 00 00 00  | ADD A, 0             | ADD A, 0                  |
| 00 00 00 01  | ADD A, 1             | ADD A, 1                  |
| 00 00 00 02  | ADD A, 2             | ADD A, 2                  |
| 00 00 00 03  | ADD A, 3             | ADD A, 3                  |
| 00 00 00 04  | ADD A, 4             | ADD A, 4                  |
| 00 00 00 05  | ADD A, 5             | ADD A, 5                  |
| 00 00 00 06  | ADD A, 6             | ADD A, 6                  |
| 00 00 00 07  | ADD A, 7             | ADD A, 7                  |
| 00 00 00 08  | ADD A, 8             | ADD A, 8                  |
| 00 00 00 09  | ADD A, 9             | ADD A, 9                  |
| 00 00 00 0A  | ADD A, 10            | ADD A, 10                 |
| 00 00 00 0B  | ADD A, 11            | ADD A, 11                 |
| 00 00 00 0C  | ADD A, 12            | ADD A, 12                 |
| 00 00 00 0D  | ADD A, 13            | ADD A, 13                 |
| 00 00 00 0E  | ADD A, 14            | ADD A, 14                 |
| 00 00 00 0F  | ADD A, 15            | ADD A, 15                 |
| 00 00 00 10  | ADD A, 16            | ADD A, 16                 |
| 00 00 00 11  | ADD A, 17            | ADD A, 17                 |
| 00 00 00 12  | ADD A, 18            | ADD A, 18                 |
| 00 00 00 13  | ADD A, 19            | ADD A, 19                 |
| 00 00 00 14  | ADD A, 20            | ADD A, 20                 |
| 00 00 00 15  | ADD A, 21            | ADD A, 21                 |
| 00 00 00 16  | ADD A, 22            | ADD A, 22                 |
| 00 00 00 17  | ADD A, 23            | ADD A, 23                 |
| 00 00 00 18  | ADD A, 24            | ADD A, 24                 |
| 00 00 00 19  | ADD A, 25            | ADD A, 25                 |
| 00 00 00 1A  | ADD A, 26            | ADD A, 26                 |
| 00 00 00 1B  | ADD A, 27            | ADD A, 27                 |
| 00 00 00 1C  | ADD A, 28            | ADD A, 28                 |
| 00 00 00 1D  | ADD A, 29            | ADD A, 29                 |
| 00 00 00 1E  | ADD A, 30            | ADD A, 30                 |
| 00 00 00 1F  | ADD A, 31            | ADD A, 31                 |
| 00 00 00 20  | ADD A, 32            | ADD A, 32                 |
| 00 00 00 21  | ADD A, 33            | ADD A, 33                 |
| 00 00 00 22  | ADD A, 34            | ADD A, 34                 |
| 00 00 00 23  | ADD A, 35            | ADD A, 35                 |
| 00 00 00 24  | ADD A, 36            | ADD A, 36                 |
| 00 00 00 25  | ADD A, 37            | ADD A, 37                 |
| 00 00 00 26  | ADD A, 38            | ADD A, 38                 |
| 00 00 00 27  | ADD A, 39            | ADD A, 39                 |
| 00 00 00 28  | ADD A, 40            | ADD A, 40                 |
| 00 00 00 29  | ADD A, 41            | ADD A, 41                 |
| 00 00 00 2A  | ADD A, 42            | ADD A, 42                 |
| 00 00 00 2B  | ADD A, 43            | ADD A, 43                 |
| 00 00 00 2C  | ADD A, 44            | ADD A, 44                 |
| 00 00 00 2D  | ADD A, 45            | ADD A, 45                 |
| 00 00 00 2E  | ADD A, 46            | ADD A, 46                 |
| 00 00 00 2F  | ADD A, 47            | ADD A, 47                 |
| 00 00 00 30  | ADD A, 48            | ADD A, 48                 |
| 00 00 00 31  | ADD A, 49            | ADD A, 49                 |
| 00 00 00 32  | ADD A, 50            | ADD A, 50                 |
| 00 00 00 33  | ADD A, 51            | ADD A, 51                 |
| 00 00 00 34  | ADD A, 52            | ADD A, 52                 |
| 00 00 00 35  | ADD A, 53            | ADD A, 53                 |
| 00 00 00 36  | ADD A, 54            | ADD A, 54                 |
| 00 00 00 37  | ADD A, 55            | ADD A, 55                 |
| 00 00 00 38  | ADD A, 56            | ADD A, 56                 |
| 00 00 00 39  | ADD A, 57            | ADD A, 57                 |
| 00 00 00 3A  | ADD A, 58            | ADD A, 58                 |
| 00 00 00 3B  | ADD A, 59            | ADD A, 59                 |
| 00 00 00 3C  | ADD A, 60            | ADD A, 60                 |
| 00 00 00 3D  | ADD A, 61            | ADD A, 61                 |
| 00 00 00 3E  | ADD A, 62            | ADD A, 62                 |
| 00 00 00 3F  | ADD A, 63            | ADD A, 63                 |
| 00 00 00 40  | ADD A, 64            | ADD A, 64                 |
| 00 00 00 41  | ADD A, 65            | ADD A, 65                 |
| 00 00 00 42  | ADD A, 66            | ADD A, 66                 |
| 00 00 00 43  | ADD A, 67            | ADD A, 67                 |
| 00 00 00 44  | ADD A, 68            | ADD A, 68                 |
| 00 00 00 45  | ADD A, 69            | ADD A, 69                 |
| 00 00 00 46  | ADD A, 70            | ADD A, 70                 |
| 00 00 00 47  | ADD A, 71            | ADD A, 71                 |
| 00 00 00 48  | ADD A, 72            | ADD A, 72                 |
| 00 00 00 49  | ADD A, 73            | ADD A, 73                 |
| 00 00 00 4A  | ADD A, 74            | ADD A, 74                 |
| 00 00 00 4B  | ADD A, 75            | ADD A, 75                 |
| 00 00 00 4C  | ADD A, 76            | ADD A, 76                 |
| 00 00 00 4D  | ADD A, 77            | ADD A, 77                 |
| 00 00 00 4E  | ADD A, 78            | ADD A, 78                 |
| 00 00 00 4F  | ADD A, 79            | ADD A, 79                 |
| 00 00 00 50  | ADD A, 80            | ADD A, 80                 |
| 00 00 00 51  | ADD A, 81            | ADD A, 81                 |
| 00 00 00 52  | ADD A, 82            | ADD A, 82                 |
| 00 00 00 53  | ADD A, 83            | ADD A, 83                 |
| 00 00 00 54  | ADD A, 84            | ADD A, 84                 |
| 00 00 00 55  | ADD A, 85            | ADD A, 85                 |
| 00 00 00 56  | ADD A, 86            | ADD A, 86                 |
| 00 00 00 57  | ADD A, 87            | ADD A, 87                 |
| 00 00 00 58  | ADD A, 88            | ADD A, 88                 |
| 00 00 00 59  | ADD A, 89            | ADD A, 89                 |
| 00 00 00 5A  | ADD A, 90            | ADD A, 90                 |
| 00 00 00 5B  | ADD A, 91            | ADD A, 91                 |
| 00 00 00 5C  | ADD A, 92            | ADD A, 92                 |
| 00 00 00 5D  | ADD A, 93            | ADD A, 93                 |
| 00 00 00 5E  | ADD A, 94            | ADD A, 94                 |
| 00 00 00 5F  | ADD A, 95            | ADD A, 95                 |
| 00 00 00 60  | ADD A, 96            | ADD A, 96                 |
| 00 00 00 61  | ADD A, 97            | ADD A, 97                 |
| 00 00 00 62  | ADD A, 98            | ADD A, 98                 |
| 00 00 00 63  | ADD A, 99            | ADD A, 99                 |
| 00 00 00 64  | ADD A, 100           | ADD A, 100                |
| 00 00 00 65  | ADD A, 101           | ADD A, 101                |
| 00 00 00 66  | ADD A, 102           | ADD A, 102                |
| 00 00 00 67  | ADD A, 103           | ADD A, 103                |
| 00 00 00 68  | ADD A, 104           | ADD A, 104                |
| 00 00 00 69  | ADD A, 105           | ADD A, 105                |
| 00 00 00 6A  | ADD A, 106           | ADD A, 106                |
| 00 00 00 6B  | ADD A, 107           | ADD A, 107                |
| 00 00 00 6C  | ADD A, 108           | ADD A, 108                |
| 00 00 00 6D  | ADD A, 109           | ADD A, 109                |
| 00 00 00 6E  | ADD A, 110           | ADD A, 110                |
| 00 00 00 6F  | ADD A, 111           | ADD A, 111                |
| 00 00 00 70  | ADD A, 112           | ADD A, 112                |
| 00 00 00 71  | ADD A, 113           | ADD A, 113                |
| 00 00 00 72  | ADD A, 114           | ADD A, 114                |
| 00 00 00 73  | ADD A, 115           | ADD A, 115                |
| 00 00 00 74  | ADD A, 116           | ADD A, 116                |
| 00 00 00 75  | ADD A, 117           | ADD A, 117                |
| 00 00 00 76  | ADD A, 118           | ADD A, 118                |
| 00 00 00 77  | ADD A, 119           | ADD A, 119                |
| 00 00 00 78  | ADD A, 120           | ADD A, 120                |
| 00 00 00 79  | ADD A, 121           | ADD A, 121                |
| 00 00 00 7A  | ADD A, 122           | ADD A, 122                |
| 00 00 00 7B  | ADD A, 123           | ADD A, 123                |
| 00 00 00 7C  | ADD A, 124           | ADD A, 124                |
| 00 00 00 7D  | ADD A, 125           | ADD A, 125                |
| 00 00 00 7E  | ADD A, 126           | ADD A, 126                |
| 00 00 00 7F  | ADD A, 127           | ADD A, 127                |
| 00 00 00 80  | ADD A, 128           | ADD A, 128                |
| 00 00 00 81  | ADD A, 129           | ADD A, 129                |
| 00 00 00 82  | ADD A, 130           | ADD A, 130                |
| 00 00 00 83  | ADD A, 131           | ADD A, 131                |
| 00 00 00 84  | ADD A, 132           | ADD A, 132                |
| 00 00 00 85  | ADD A, 133           | ADD A, 133                |
| 00 00 00 86  | ADD A, 134           | ADD A, 134                |
| 00 00 00 87  | ADD A, 135           | ADD A, 135                |
| 00 00 00 88  | ADD A, 136           | ADD A, 136                |
| 00 00 00 89  | ADD A, 137           | ADD A, 137                |
| 00 00 00 8A  | ADD A, 138           | ADD A, 138                |
| 00 00 00 8B  | ADD A, 139           | ADD A, 139                |
| 00 00 00 8C  | ADD A, 140           | ADD A, 140                |
| 00 00 00 8D  | ADD A, 141           | ADD A, 141                |
| 00 00 00 8E  | ADD A, 142           | ADD A, 142                |
| 00 00 00 8F  | ADD A, 143           | ADD A, 143                |
| 00 00 00 90  | ADD A, 144           | ADD A, 144                |
| 00 00 00 91  | ADD A, 145           | ADD A, 145                |
| 00 00 00 92  | ADD A, 146           | ADD A, 146                |
| 00 00 00 93  | ADD A, 147           | ADD A, 147                |
| 00 00 00 94  | ADD A, 148           | ADD A, 148                |
| 00 00 00 95  | ADD A, 149           | ADD A, 149                |
| 00 00 00 96  | ADD A, 150           | ADD A, 150                |
| 00 00 00 97  | ADD A, 151           | ADD A, 151                |
| 00 00 00 98  | ADD A, 152           | ADD A, 152                |
| 00 00 00 99  | ADD A, 153           | ADD A, 153                |
| 00 00 00 9A  | ADD A, 154           | ADD A, 154                |
| 00 00 00 9B  | ADD A, 155           | ADD A, 155                |
| 00 00 00 9C  | ADD A, 156           | ADD A, 156                |
| 00 00 00 9D  | ADD A, 157           | ADD A, 157                |
| 00 00 00 9E  | ADD A, 158           | ADD A, 158                |
| 00 00 00 9F  | ADD A, 159           | ADD A, 159                |
| 00 00 00 A0  | ADD A, 160           | ADD A, 160                |
| 00 00 00 A1  | ADD A, 161           | ADD A, 161                |
| 00 00 00 A2  | ADD A, 162           | ADD A, 162                |
| 00 00 00 A3  | ADD A, 163           | ADD A, 163                |
| 00 00 00 A4  | ADD A, 164           | ADD A, 164                |
| 00 00 00 A5  | ADD A, 165           | ADD A, 165                |
| 00 00 00 A6  | ADD A, 166           | ADD A, 166                |
| 00 00 00 A7  | ADD A, 167           | ADD A, 167                |
| 00 00 00 A8  | ADD A, 168           | ADD A, 168                |
| 00 00 00 A9  | ADD A, 169           | ADD A, 169                |
| 00 00 00 AA  | ADD A, 170           | ADD A, 170                |
| 00 00 00 AB  | ADD A, 171           | ADD A, 171                |
| 00 00 00 AC  | ADD A, 172           | ADD A, 172                |
| 00 00 00 AD  | ADD A, 173           | ADD A, 173                |
| 00 00 00 AE  | ADD A, 174           | ADD A, 174                |
| 00 00 00 AF  | ADD A, 175           | ADD A, 175                |
| 00 00 00 B0  | ADD A, 176           | ADD A, 176                |
| 00 00 00 B1  | ADD A, 177           | ADD A, 177                |
| 00 00 00 B2  | ADD A, 178           | ADD A, 178                |
| 00 00 00 B3  | ADD A, 179           | ADD A, 179                |
| 00 00 00 B4  | ADD A, 180           | ADD A, 180                |
| 00 00 00 B5  | ADD A, 181           | ADD A, 181                |
| 00 00 00 B6  | ADD A, 182           | ADD A, 182                |
| 00 00 00 B7  | ADD A, 183           | ADD A, 183                |
| 00 00 00 B8  | ADD A, 184           | ADD A, 184                |
| 00 00 00 B9  | ADD A, 185           | ADD A, 185                |
| 00 00 00 BA  | ADD A, 186           | ADD A, 186                |
| 00 00 00 BB  | ADD A, 187           | ADD A, 187                |
| 00 00 00 BC  | ADD A, 188           | ADD A, 188                |
| 00 00 00 BD  | ADD A, 189           | ADD A, 189                |
| 00 00 00 BE  | ADD A, 190           | ADD A, 190                |
| 00 00 00 BF  | ADD A, 191           | ADD A, 191                |
| 00 00 00 C0  | ADD A, 192           | ADD A, 192                |
| 00 00 00 C1  | ADD A, 193           | ADD A, 193                |
| 00 00 00 C2  | ADD A, 194           | ADD A, 194                |
| 00 00 00 C3  | ADD A, 195           | ADD A, 195                |
| 00 00 00 C4  | ADD A, 196           | ADD A, 196                |
| 00 00 00 C5  | ADD A, 197           | ADD A, 197                |
| 00 00 00 C6  | ADD A, 198           | ADD A, 198                |
| 00 00 00 C7  | ADD A, 199           | ADD A, 199                |
| 00 00 00 C8  | ADD A, 200           | ADD A, 200                |
| 00 00 00 C9  | ADD A, 201           | ADD A, 201                |
| 00 00 00 CA  | ADD A, 202           | ADD A, 202                |
| 00 00 00 CB  | ADD A, 203           | ADD A, 203                |
| 00 00 00 CC  | ADD A, 204           | ADD A, 204                |
| 00 00 00 CD  | ADD A, 205           | ADD A, 205                |
| 00 00 00 CE  | ADD A, 206           | ADD A, 206                |
| 00 00 00 CF  | ADD A, 207           | ADD A, 207                |
| 00 00 00 D0  | ADD A, 208           | ADD A, 208                |
| 00 00 00 D1  | ADD A, 209           | ADD A, 209                |
| 00 00 00 D2  | ADD A, 210           | ADD A, 210                |
| 00 00 00 D3  | ADD A, 211           | ADD A, 211                |
| 00 00 00 D4  | ADD A, 212           | ADD A, 212                |
| 00 00 00 D5  | ADD A, 213           | ADD A, 213                |
| 00 00 00 D6  | ADD A, 214           | ADD A, 214                |
| 00 00 00 D7  | ADD A, 215           | ADD A, 215                |
| 00 00 00 D8  | ADD A, 216           | ADD A, 216                |
| 00 00 00 D9  | ADD A, 217           | ADD A, 217                |
| 00 00 00 DA  | ADD A, 218           | ADD A, 218                |
| 00 00 00 DB  | ADD A, 219           | ADD A, 219                |
| 00 00 00 DC  | ADD A, 220           | ADD A, 220                |
| 00 00 00 DD  | ADD A, 221           | ADD A, 221                |
| 00 00 00 DE  | ADD A, 222           | ADD A, 222                |
| 00 00 00 DF  | ADD A, 223           | ADD A, 223                |
| 00 00 00 E0  | ADD A, 224           | ADD A, 224                |
| 00 00 00 E1  | ADD A, 225           | ADD A, 225                |
| 00 00 00 E2  | ADD A, 226           | ADD A, 226                |
| 00 00 00 E3  | ADD A, 227           | ADD A, 227                |
| 00 00 00 E4  | ADD A, 228           | ADD A, 228                |
| 00 00 00 E5  | ADD A, 229           | ADD A, 229                |
| 00 00 00 E6  | ADD A, 230           | ADD A, 230                |
| 00 00 00 E7  | ADD A, 231           | ADD A, 231                |
| 00 00 00 E8  | ADD A, 232           | ADD A, 232                |
| 00 00 00 E9  | ADD A, 233           | ADD A, 233                |
| 00 00 00 EA  | ADD A, 234           | ADD A, 234                |
| 00 00 00 EB  | ADD A, 235           | ADD A, 235                |
| 00 00 00 EC  | ADD A, 236           | ADD A, 236                |
| 00 00 00 ED  | ADD A, 237           | ADD A, 237                |
| 00 00 00 EE  | ADD A, 238           | ADD A, 238                |
| 00 00 00 EF  | ADD A, 239           | ADD A, 239                |
| 00 00 00 F0  | ADD A, 240           | ADD A, 240                |
| 00 00 00 F1  | ADD A, 241           | ADD A, 241                |
| 00 00 00 F2  | ADD A, 242           | ADD A, 242                |
| 00 00 00 F3  | ADD A, 243           | ADD A, 243                |
| 00 00 00 F4  | ADD A, 244           | ADD A, 244                |
| 00 00 00 F5  | ADD A, 245           | ADD A, 245                |
| 00 00 00 F6  | ADD A, 246           | ADD A, 246                |
| 00 00 00 F7  | ADD A, 247           | ADD A, 247                |
| 00 00 00 F8  | ADD A, 248           | ADD A, 248                |
| 00 00 00 F9  | ADD A, 249           | ADD A, 249                |
| 00 00 00 FA  | ADD A, 250           | ADD A, 250                |
| 00 00 00 FB  | ADD A, 251           | ADD A, 251                |
| 00 00 00 FC  | ADD A, 252           | ADD A, 252                |
| 00 00 00 FD  | ADD A, 253           | ADD A, 253                |
| 00 00 00 FE  | ADD A, 254           | ADD A, 254                |
| 00 00 00 FF  | ADD A, 255           | ADD A, 255                |



# MCS-48 INSTRUCTION SET

## 8048H/8748H/8049H/8749H INSTRUCTION SET SUMMARY

| Mnemonic            | Description                   | Bytes | Cycle |
|---------------------|-------------------------------|-------|-------|
| <b>Accumulator</b>  |                               |       |       |
| ADD A, R            | Add register to A             | 1     | 1     |
| ADD A, @R           | Add data memory to A          | 1     | 1     |
| ADD A, #data        | Add immediate to A            | 2     | 2     |
| ADDC A, R           | Add register with carry       | 1     | 1     |
| ADDC A, @R          | Add data memory with carry    | 1     | 1     |
| ADDC A, #data       | Add immediate with carry      | 2     | 2     |
| ANL A, R            | And register to A             | 1     | 1     |
| ANL A, @R           | And data memory to A          | 1     | 1     |
| ANL A, #data        | And immediate to A            | 2     | 2     |
| ORL A, R            | Or register to A              | 1     | 1     |
| ORL A, @R           | Or data memory to A           | 1     | 1     |
| ORL A, #data        | Or immediate to A             | 2     | 2     |
| XRL A, R            | Exclusive Or register to A    | 1     | 1     |
| XRL A, @R           | Exclusive or data memory to A | 1     | 1     |
| XRL A, #data        | Exclusive or immediate to A   | 2     | 2     |
| INC A               | Increment A                   | 1     | 1     |
| DEC A               | Decrement A                   | 1     | 1     |
| CLR A               | Clear A                       | 1     | 1     |
| CPL A               | Complement A                  | 1     | 1     |
| DA A                | Decimal Adjust A              | 1     | 1     |
| SWAP A              | Swap nibbles of A             | 1     | 1     |
| RL A                | Rotate A left                 | 1     | 1     |
| RLC A               | Rotate A left through carry   | 1     | 1     |
| RR A                | Rotate A right                | 1     | 1     |
| RRC A               | Rotate A right through carry  | 1     | 1     |
| <b>Input/Output</b> |                               |       |       |
| IN A, P             | Input port to A               | 1     | 2     |
| OUTL P, A           | Output A to port              | 1     | 2     |
| ANL P, #data        | And immediate to port         | 2     | 2     |
| ORL P, #data        | Or immediate to port          | 2     | 2     |
| INS A, BUS          | Input BUS to A                | 1     | 2     |
| OUTL BUS, A         | Output A to BUS               | 1     | 2     |
| ANL BUS, #data      | And immediate to BUS          | 2     | 2     |
| ORL BUS, #data      | Or immediate to BUS           | 2     | 2     |
| MOVD A, P           | Input Expander port to A      | 1     | 2     |
| MOVD P, A           | Output A to Expander port     | 1     | 2     |
| ANLD P, A           | And A to Expander port        | 1     | 2     |
| ORLD P, A           | Or A to Expander port         | 1     | 2     |
| <b>Registers</b>    |                               |       |       |
| INC R               | Increment register            | 1     | 1     |
| INC @R              | Increment data memory         | 1     | 1     |
| DEC R               | Decrement register            | 1     | 1     |
| <b>Branch</b>       |                               |       |       |
| JMP addr            | Jump unconditional            | 2     | 2     |
| JMPP @A             | Jump indirect                 | 1     | 2     |
| DJNZ R, addr        | Decrement register and jump   | 2     | 2     |
| JC addr             | Jump on Carry = 1             | 2     | 2     |
| JNC addr            | Jump on Carry = 0             | 2     | 2     |
| JZ addr             | Jump on A Zero                | 2     | 2     |
| JNZ addr            | Jump on A not Zero            | 2     | 2     |
| JTO addr            | Jump on T0 = 1                | 2     | 2     |
| JNT0 addr           | Jump on T0 = 0                | 2     | 2     |
| JT1 addr            | Jump on T1 = 1                | 2     | 2     |
| JNT1 addr           | Jump on T1 = 0                | 2     | 2     |
| JF0 addr            | Jump on F0 = 1                | 2     | 2     |
| JF1 addr            | Jump on F1 = 1                | 2     | 2     |
| JTF addr            | Jump on timer flag = 1        | 2     | 2     |
| JNI addr            | Jump on INT = 0               | 2     | 2     |
| JBb addr            | Jump on Accumulator Bit       | 2     | 2     |

| Mnemonic             | Description                       | Bytes | Cycles |
|----------------------|-----------------------------------|-------|--------|
| <b>Subroutine</b>    |                                   |       |        |
| CALL addr            | Jump to subroutine                | 2     | 2      |
| RET                  | Return                            | 1     | 2      |
| RETR                 | Return and restore status         | 1     | 2      |
| <b>Flags</b>         |                                   |       |        |
| CLR C                | Clear Carry                       | 1     | 1      |
| CPL C                | Complement Carry                  | 1     | 1      |
| CLR F0               | Clear Flag 0                      | 1     | 1      |
| CPL F0               | Complement Flag 0                 | 1     | 1      |
| CLR F1               | Clear Flag 1                      | 1     | 1      |
| CPL F1               | Complement Flag 1                 | 1     | 1      |
| <b>Data Moves</b>    |                                   |       |        |
| MOV A, R             | Move register to A                | 1     | 1      |
| MOV A, @R            | Move data memory to A             | 1     | 1      |
| MOV A, #data         | Move immediate to A               | 2     | 2      |
| MOV R, A             | Move A to register                | 1     | 1      |
| MOV @R, A            | Move A to data memory             | 1     | 1      |
| MOV R, #data         | Move immediate to register        | 2     | 2      |
| MOV @R, #data        | Move immediate to data memory     | 2     | 2      |
| MOV A, PSW           | Move PSW to A                     | 1     | 1      |
| MOV PSW, A           | Move A to PSW                     | 1     | 1      |
| XCH A, R             | Exchange A and register           | 1     | 1      |
| XCH A, @R            | Exchange A and data memory        | 1     | 1      |
| XCHD A, @R           | Exchange nibble of A and register | 1     | 1      |
| MOVX A, @R           | Move external data memory to A    | 1     | 2      |
| MOVX @R, A           | Move A to external data memory    | 1     | 2      |
| MOVP A, @A           | Move to A from current page       | 1     | 2      |
| MOVP3 A, @A          | Move to A from Page 3             | 1     | 2      |
| <b>Timer/Counter</b> |                                   |       |        |
| MOV A, T             | Read Timer/Counter                | 1     | 1      |
| MOV T, A             | Load Timer/Counter                | 1     | 1      |
| STRT T               | Start Timer                       | 1     | 1      |
| STRT CNT             | Start Counter                     | 1     | 1      |
| STOP TCNT            | Stop Timer/Counter                | 1     | 1      |
| EN TCNTI             | Enable Timer/Counter Interrupt    | 1     | 1      |
| DIS TCNTI            | Disable Timer/Counter Interrupt   | 1     | 1      |
| <b>Control</b>       |                                   |       |        |
| EN I                 | Enable external interrupt         | 1     | 1      |
| DIS I                | Disable external interrupt        | 1     | 1      |
| SEL RB0              | Select register bank 0            | 1     | 1      |
| SEL RB1              | Select register bank 1            | 1     | 1      |
| SEL MB0              | Select memory bank 0              | 1     | 1      |
| SEL MB1              | Select memory bank 1              | 1     | 1      |
| ENT0 CLK             | Enable Clock output on T0         | 1     | 1      |
| <b>NOP</b>           |                                   |       |        |
| NOP                  | No Operation                      | 1     | 1      |

Mnemonics copyright Intel Corporation 1982.



# MCS-48 INSTRUCTION SET

## 8021H/8020H INSTRUCTION SET SUMMARY

| Mnemonic            | Description                   | Bytes | Cycle |
|---------------------|-------------------------------|-------|-------|
| <b>Accumulator</b>  |                               |       |       |
| ADD A,R             | Add register to A             | 1     | 1     |
| ADD A,@R            | Add data memory to A          | 1     | 1     |
| ADD A,#data         | Add immediate to A            | 2     | 2     |
| ADDC A,R            | Add with carry                | 1     | 1     |
| ADDC A,@R           | Add with carry                | 1     | 1     |
| ADDC A,#data        | Add with carry                | 2     | 2     |
| ANL A,R             | And register to A             | 1     | 1     |
| ANL A,@R            | And data memory to A          | 1     | 1     |
| ANL A,#data         | And immediate to A            | 2     | 2     |
| ORL A,R             | Or register to A              | 1     | 1     |
| ORL A,@R            | Or data memory to A           | 1     | 1     |
| ORL A,#data         | Or immediate to A             | 2     | 2     |
| XRL A,R             | Exclusive Or register to A    | 1     | 1     |
| XRL A,@R            | Exclusive or data memory to A | 1     | 1     |
| XRL A,#data         | Exclusive or immediate to A   | 2     | 2     |
| INC A               | Increment A                   | 1     | 1     |
| DEC A               | Decrement A                   | 1     | 1     |
| CLR A               | Clear A                       | 1     | 1     |
| CPL A               | Complement A                  | 1     | 1     |
| DA A                | Decimal Adjust A              | 1     | 1     |
| SWAP A              | Swap nibbles of A             | 1     | 1     |
| RL A                | Rotate A left                 | 1     | 1     |
| RLC A               | Rotate A left through carry   | 1     | 1     |
| RR A                | Rotate A right                | 1     | 1     |
| RRC A               | Rotate A right through carry  | 1     | 1     |
| <b>Input/Output</b> |                               |       |       |
| IN A,P              | Input port to A               | 1     | 2     |
| OUTL P,A            | Output A to port              | 1     | 2     |
| *MOVD A,P           | Input Expander port to A      | 1     | 2     |
| *MOVD P,A           | Output A to Expander port     | 1     | 2     |
| *ANLD P,A           | And A to Expander port        | 1     | 2     |
| *ORLD P,A           | Or A to Expander port         | 1     | 2     |
| <b>Registers</b>    |                               |       |       |
| INC R               | Increment register            | 1     | 1     |
| INC @R              | Increment data memory         | 1     | 1     |

| Mnemonic             | Description                               | Bytes | Cycle |
|----------------------|---|-------|-------|
| <b>Branch</b>        |   |       |       |
| JMP addr             | Jump unconditional                        | 2     | 2     |
| JMPP @A              | Jump indirect                             | 1     | 2     |
| DJNZ R,addr          | Decrement register and Jump on R not zero | 2     | 2     |
| JC addr              | Jump on Carry = 1                         | 2     | 2     |
| JNC addr             | Jump on Carry = 0                         | 2     | 2     |
| JZ addr              | Jump on A Zero                            | 2     | 2     |
| JNZ addr             | Jump on A not Zero                        | 2     | 2     |
| JT1 addr             | Jump on T1 = 1                            | 2     | 2     |
| JNT1 addr            | Jump on T1 = 0                            | 2     | 2     |
| JTF addr             | Jump on timer flag                        | 2     | 2     |
| <b>Subroutine</b>    |   |       |       |
| CALL addr            | Jump to subroutine                        | 2     | 2     |
| RET                  | Return                                    | 1     | 2     |
| <b>Flags</b>         |   |       |       |
| CLR C                | Clear Carry                               | 1     | 1     |
| CPL C                | Complement Carry                          | 1     | 1     |
| <b>Data Moves</b>    |   |       |       |
| MOV A,R              | Move register to A                        | 1     | 1     |
| MOV A,@R             | Move data memory to A                     | 1     | 1     |
| MOV A,#data          | Move immediate to A                       | 2     | 2     |
| MOV R,A              | Move A to register                        | 1     | 1     |
| MOV @R,A             | Move A to data memory                     | 1     | 1     |
| MOV R,#data          | Move immediate to register                | 2     | 2     |
| MOV @R,#data         | Move immediate to data memory             | 2     | 2     |
| XCH A,R              | Exchange A and register                   | 1     | 1     |
| XCH A,@R             | Exchange A and data memory                | 1     | 2     |
| XCHD A,@R            | Exchange nibble of A and register         | 1     | 1     |
| MOVP A,@A            | Move to A from current page               | 1     | 2     |
| <b>Timer/Counter</b> |   |       |       |
| MOV A,T              | Read Timer/Counter                        | 1     | 1     |
| MOV T,A              | Load Timer/Counter                        | 1     | 1     |
| STRT T               | Start Timer                               | 1     | 1     |
| STRT CNT             | Start Counter                             | 1     | 1     |
| STOP TCNT            | Stop Timer/Counter                        | 1     | 1     |
| NOP                  | No Operation                              | 1     | 1     |

**Instruction Set** — The following instructions, which are found in the 8748, have been deleted from the 8021H instruction set.

| Data Moves | Registers    | Branch    | Timer             | Control  | Input/Output  |
|------------|--------------|-----------|-------------------|----------|---------------|
| MOV A,PSW  | DEC R        | JTO addr  | EN TCNT1          | EN I     | ANL P,#data   |
| MOV PSW,A  |              | JNTO addr | DIS TCNT1         | DIS I    | ORL P,#data   |
| MOVX A,@R  | <b>Flags</b> | JFO addr  |                   | SEL R80  | INS A,BUS *   |
| MOVX @R,A  | CLR F0       | JF1 addr  | <b>Subroutine</b> | SEL R81  | OUTL BUS,A *  |
| MOV3 A,@A  | CPL F0       | JNI addr  | RETR              | SEL M80  | ANL BUS,#data |
|            | CLR F1       | JBb addr  |                   | SEL M81  | ORL BUS,#data |
|            | CPL F1       |           |                   | ENTO CLK |               |

\* These Instructions have been replaced in the 8021H by IN A,PO and OUTL PO,A respectively.

\*Not included in 8020H.

Mnemonics copyright Intel Corp. 1981.



# 8022H INSTRUCTION SET SUMMARY

| Mnemonic               | Description                               | Bytes | Cycle | Hexadecimal Opcode      |
|------------------------|---|-------|-------|-------------------------|
| <b>Accumulator</b>     |   |       |       |                         |
| ADD A,R <sub>r</sub>   | Add register to A                         | 1     | 1     | 68-6F                   |
| ADD A,@R               | Add data memory to A                      | 1     | 1     | 60-61                   |
| ADD A,#data            | Add immediate to A                        | 2     | 2     | 03                      |
| ADDC A,R <sub>r</sub>  | Add register with carry                   | 1     | 1     | 78-7F                   |
| ADDC A,@R              | Add data memory with carry                | 1     | 1     | 70-71                   |
| ADDC A,#data           | Add immediate with carry                  | 2     | 2     | 13                      |
| ANL A,R <sub>r</sub>   | And register to A                         | 1     | 1     | 58-5F                   |
| ANL A,@R               | And data memory to A                      | 1     | 1     | 50-51                   |
| ANL A,#data            | And immediate to A                        | 2     | 2     | 53                      |
| ORL A,R <sub>r</sub>   | Or register to A                          | 1     | 1     | 48-4F                   |
| ORL A,@R               | Or data memory to A                       | 1     | 1     | 40-41                   |
| ORL A,#data            | Or immediate to A                         | 2     | 2     | 43                      |
| XRL A,R <sub>r</sub>   | Exclusive Or register to A                | 1     | 1     | 08-0F                   |
| XRL A,@R               | Exclusive Or data memory to A             | 1     | 1     | D0-D1                   |
| XRL A,#data            | Exclusive Or immediate to A               | 2     | 2     | 03                      |
| INC A                  | Increment A                               | 1     | 1     | 17                      |
| DEC A                  | Decrement A                               | 1     | 1     | 07                      |
| CLR A                  | Clear A                                   | 1     | 1     | 27                      |
| CPL A                  | Complement A                              | 1     | 1     | 37                      |
| DA A                   | Decimal adjust A                          | 1     | 1     | 57                      |
| SWAP A                 | Swap nibbles of A                         | 1     | 1     | 47                      |
| RL A                   | Rotate A left                             | 1     | 1     | E7                      |
| RLC A                  | Rotate A left through carry               | 1     | 1     | F7                      |
| RR A                   | Rotate A right                            | 1     | 1     | 77                      |
| RRC A                  | Rotate A right through carry              | 1     | 1     | 67                      |
| <b>Input / Output</b>  |   |       |       |                         |
| IN A,P <sub>p</sub>    | Input port to A                           | 1     | 2     | 08,09,0A                |
| OUTL P <sub>p</sub> ,A | Output A to port                          | 1     | 2     | 90,99,3A                |
| MOVD A,P <sub>p</sub>  | Input expander port to A                  | 1     | 2     | 0C-0F                   |
| MOVD P <sub>p</sub> ,A | Output A to expander port                 | 1     | 2     | 3C-3F                   |
| ANLD P <sub>p</sub> ,A | And A to expander port                    | 1     | 2     | 9C-9F                   |
| ORLD P <sub>p</sub> ,A | Or A to expander port                     | 1     | 2     | 8C-8F                   |
| <b>Registers</b>       |   |       |       |                         |
| INC R <sub>r</sub>     | Increment register                        | 1     | 1     | 18-1F                   |
| INC @R                 | Increment data memory                     | 1     | 1     | 10-11                   |
| <b>Branch</b>          |   |       |       |                         |
| JMP addr               | Jump unconditional                        | 2     | 2     | 04,24,44,64,84,A4,C4,E4 |
| JMPP @A                | Jump indirect                             | 1     | 2     | B3                      |
| DJNZ R,addr            | Decrement register and jump on R not zero | 2     | 2     | E8-EF                   |
| JC addr                | Jump on carry=1                           | 2     | 2     | F6                      |
| JNC addr               | Jump on carry=0                           | 2     | 2     | E6                      |
| JZ addr                | Jump on A zero                            | 2     | 2     | C6                      |
| JNZ addr               | Jump on A not zero                        | 2     | 2     | 96                      |

Mnemonics copyright Intel Corporation 1982.

| Mnemonic                  | Description                          | Bytes | Cycle | Hexadecimal Opcode      |
|---------------------------|--------------------------------------|-------|-------|-------------------------|
| JT0                       | Jump on T0=1                         | 2     | 2     | 36                      |
| JNT0                      | Jump on T0=0                         | 2     | 2     | 26                      |
| JT1 addr                  | Jump on T1=1                         | 2     | 2     | 56                      |
| JNT1 addr                 | Jump on T1=0                         | 2     | 2     | 46                      |
| JTF addr                  | Jump on timer flag                   | 2     | 2     | 16                      |
| <b>Subroutine</b>         |                                      |       |       |                         |
| CALL addr                 | Jump to subroutine                   | 1     | 2     | 14,34,54,74,94,B4,D4,F4 |
| RET                       | Return                               | 1     | 2     | 83                      |
| <b>Flags</b>              |                                      |       |       |                         |
| CLR C                     | Clear carry                          | 1     | 1k    | 97                      |
| CPL C                     | Complement carry                     | 1     | 1     | A7                      |
| <b>Data Moves</b>         |                                      |       |       |                         |
| MOV A,R <sub>r</sub>      | Move register to A                   | 1     | 1     | F8-FF                   |
| MOV A,@R                  | Move data memory to A                | 1     | 1     | F0-F1                   |
| MOV A,#data               | Move immediate to A                  | 2     | 2     | 23                      |
| MOV R <sub>r</sub> ,A     | Move A to register                   | 1     | 1     | A8-AF                   |
| MOV @R,A                  | Move A to data memory                | 1     | 1     | A0-A1                   |
| MOV R <sub>r</sub> ,#data | Move immediate to register           | 2     | 2     | B8-BF                   |
| MOV @R,#data              | Move immediate to data memory        | 2     | 2     | BO-B1                   |
| XCH A,R <sub>r</sub>      | Exchange A and register              | 1     | 1     | 28-2F                   |
| XCH A,@R                  | Exchange A and data memory           | 1     | 1     | 20-21                   |
| XCHD a,@R                 | Exchange nibble of A and register    | 1     | 1     | 30-31                   |
| MOVP A,@A                 | Move to A from current page          | 1     | 2     | A3                      |
| <b>Timer / Counter</b>    |                                      |       |       |                         |
| MOV A,T                   | Read timer / counter                 | 1     | 1     | 42                      |
| MOV T,A                   | Load timer / counter                 | 1     | 1     | 62                      |
| STRT T                    | Start timer                          | 1     | 1     | 55                      |
| STRT CNT                  | Start counter                        | 1     | 1     | 45                      |
| STOP TCNT                 | Stop timer / counter                 | 1     | 1     | 65                      |
| <b>A / D Converter</b>    |                                      |       |       |                         |
| RAD                       | Move conversion result register to A | 1     | 2     | 80                      |
| SEL AN0                   | Select analog input zero             | 1     | 1     | 85                      |
| SEL AN1                   | Select analog input one              | 1     | 1     | 95                      |
| <b>Interrupts</b>         |                                      |       |       |                         |
| EN I                      | Enable external interrupt            | 1     | 1     | 05                      |
| DIS I                     | Disable external interrupt           | 1     | 1     | 15                      |
| EN TCNTI                  | Enable timer / counter interrupt     | 1     | 1     | 25                      |
| DIS TCNTI                 | Disable timer / counter interrupt    | 1     | 1     | 35                      |
| RET I                     | Return from interrupt                | 1     | 2     | 93                      |
| NOP                       | No operation                         | 1     | 1     | 00                      |

**Instruction Set** — The following instructions, which are found in the 8748, have been deleted from the 8022H instruction set.

| Data Moves | Registers    | Branch   | Control  | Input/Output  |
|------------|--------------|----------|----------|---------------|
| MOV A,PSW  | DEC R        |          |          | ANL P,#data   |
| MOV PSW,A  |              |          |          | ORL P,#data   |
| MOVX A,@R  | <b>Flags</b> | JF0 addr | SEL R80  | INS A,BUS *   |
| MOVX @R,A  | JF1          | JF1 addr | SEL R81  | OUTL BUS,A *  |
| MOV3 A,@A  | CPL F0       | JN1 addr | SEL M80  | ANL BUS,#data |
|            | CPL F1       | JBb addr | SEL M81  | ORL BUS,#data |
|            | CPL F1       |          | ENT0 CLK |               |
|            |              |          | RETR     |               |

\*These instructions have been replaced in the 8022H by IN A,PO and OUTL PO,A respectively.



## MCS-48 Instruction Set

### SYMBOLS AND ABBREVIATIONS USED

|        |                                       |
|--------|---------------------------------------|
| A      | Accumulator                           |
| AC     | Auxiliary Carry                       |
| addr   | 12-Bit Program Memory Address         |
| Bb     | Bit Designator (b = 0-7)              |
| BS     | Bank Switch                           |
| BUS    | BUS Port                              |
| C      | Carry                                 |
| CLK    | Clock                                 |
| CNT    | Event Counter                         |
| CRR    | Conversion Result Register            |
| D      | Mnemonic for 4-Bit Digit (Nibble)     |
| data   | 8-Bit Number or Expression            |
| DBF    | Memory Bank Flip-Flop                 |
| F0, F1 | Flag 0, Flag 1                        |
| I      | Interrupt                             |
| P      | Mnemonic for "in-page" Operation      |
| PC     | Program Counter                       |
| Pp     | Port Designator (p = 1, 2 or 4-7)     |
| PSW    | Program Status Word                   |
| Rr     | Register Designator (r = 0, 1 or 0-7) |
| SP     | Stack Pointer                         |
| T      | Timer                                 |
| TF     | Timer Flag                            |
| T0, T1 | Test 0, Test 1                        |
| X      | Mnemonic for External RAM             |
| #      | Immediate Data Prefix                 |
| @      | Indirect Address Prefix               |
| \$     | Current Value of Program Counter      |
| (X)    | Contents of X                         |
| ((X))  | Contents of Location Addressed by X   |
| ←      | Is Replaced by                        |

Mnemonics copyright Intel Corporation 1982.



## MCS-48 INSTRUCTION SET

### ADD A,R<sub>r</sub> Add Register Contents to Accumulator

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 1 | r | r | r |
|---|---|---|---|

 68H-6FH

**Description:** The contents of register 'r' are added to the accumulator. Carry is affected.

**Operation:**  $(A) \leftarrow (A) + (R_r)$  r = 0-7

**Example:** ADDREG: ADD A,R6 ;ADD REG 6 CONTENTS  
;TO ACC

### ADD A,@R<sub>r</sub> Add Data Memory Contents to Accumulator

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 0 | r |
|---|---|---|---|

 60H-61H

**Description:** The contents of the resident data memory location addressed by register 'r' bits 0-5\* are added to the accumulator. Carry is affected.

**Operation:**  $(A) \leftarrow (A) + ((R_r))$  R = 0-1

**Example:** ADDM: MOV R0, #01FH ;MOVE '1F' HEX TO REG 0  
ADD A, @R0 ;ADD VALUE OF LOCATION  
;31 TO ACC

### ADD A,#data Add Immediate Data to Accumulator

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
|---|---|---|---|

|    |    |    |    |
|----|----|----|----|
| d7 | d6 | d5 | d4 |
|----|----|----|----|

|    |    |    |    |
|----|----|----|----|
| d3 | d2 | d1 | d0 |
|----|----|----|----|

 03H

**Description:** This is a 2-cycle instruction. The specified data is added to the accumulator. Carry is affected.

**Operation:**  $(A) \leftarrow (A) + \text{data}$

**Example:** ADDID: ADD A,#ADDER: ;ADD VALUE OF SYMBOL  
;'ADDER' TO ACC

### ADDC A,R<sub>r</sub> Add Carry and Register Contents to Accumulator

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 1 | r | r | r |
|---|---|---|---|

 78H-7FH

**Description:** The content of the carry bit is added to accumulator location 0 and the carry bit cleared. The contents of register 'r' are then added to the accumulator. Carry is affected.

**Operation:**  $(A) \leftarrow (A) + (R_r) + (C)$  R = 0-7

**Example:** ADDRGC: ADDC A,R4 ;ADD CARRY AND REG 4  
;CONTENTS TO ACC



## MCS-48 INSTRUCTION SET

### ADDC A,@R<sub>r</sub> Add Carry and Data Memory Contents to Accumulator

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 0 | r |
|---|---|---|---|

 70H-71H

**Description:** The content of the carry bit is added to accumulator location 0 and the carry bit cleared. Then the contents of the resident data memory location addressed by register 'r' bits 0-5\* are added to the accumulator. Carry is affected.

**Operation:**  $(A) \leftarrow (A) + ((R_r)) + (C)$  R = 0-1

**Example:** ADDMC: MOV R1,#40 ;MOVE '40' DEC TO REG 1  
                   ADDC A,@R1 ;ADD CARRY AND LOCATION 40  
                                   ;CONTENTS TO ACC

### ADDC A,#data Add Carry and Immediate Data to Accumulator

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
|---|---|---|---|

|    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|
| d7 | d6 | d5 | d4 | d3 | d2 | d1 | d0 |
|----|----|----|----|----|----|----|----|

 13H

**Description:** This is a 2-cycle instruction. The content of the carry bit is added to accumulator location 0 and the carry bit cleared. Then the specified data is added to the accumulator. Carry is affected.

**Operation:**  $(A) \leftarrow (A) + \text{data} + (C)$

**Example:** ADDC A,#225 ;ADD CARRY AND '225' DEC  
                                   ;TO ACC

### ANL A,R<sub>r</sub> Logical AND Accumulator with Register Mask

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 1 | r | r | r |
|---|---|---|---|

 58H-5FH

**Description:** Data in the accumulator is logically ANDed with the mask contained in working register 'r'.

**Operation:**  $(A) \leftarrow (A) \text{ AND } (R_r)$  R = 0-7

**Example:** ANDREG: ANL A,R3 ;'AND' ACC CONTENTS WITH MASK  
                                   ;IN REG 3

### ANL A,@R<sub>r</sub> Logical AND Accumulator with Memory Mask

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 0 | r |
|---|---|---|---|

 50H-51H

**Description:** Data in the accumulator is logically ANDed with the mask contained in the data memory location referenced by register 'r' bits 0-5\*.

**Operation:**  $(A) \leftarrow (A) \text{ AND } ((R_r))$  r = 0-1

**Example:** ANDDM: MOV R0,#03FH ;MOVE '3F' HEX TO REG 0  
                   ANL A, @R0 ;'AND' ACC CONTENTS WITH  
                                   ;MASK IN LOCATION 63



**ANL A,#data** Logical AND Accumulator with Immediate Mask

**Encoding:**    0 1 0 1   0 0 1 1    d7 d6 d5 d4   d3 d2 d1 d0    53H

**Operation:**  $(A) \leftarrow (A) \text{ AND data}$

ANL A,#3 + X/Y ;'AND' ACC CONTENTS

 $3 + X/Y$ 

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 0 | 0 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
|---|---|---|---|

|    |    |    |    |
|----|----|----|----|
| d7 | d6 | d5 | d4 |
|----|----|----|----|

|    |    |    |    |
|----|----|----|----|
| d3 | d2 | d1 | d0 |
|----|----|----|----|

 98H

**Operation:** (BUS) ← (BUS) AND data

**Example:** ANDBUS: ANL BUS,#MASK ;'AND' BUS CONTENTS  
;WITH MASK EQUAL VALUE  
;OF SYMBOL 'MASK'

**Encoding:**

|         |         |
|---------|---------|
| 1 0 0 1 | 1 0 p p |
|---------|---------|

|             |             |
|-------------|-------------|
| d7 d6 d5 d4 | d3 d2 d1 d0 |
|-------------|-------------|

 99H-9AH

**Operation:**  $(P_p) \leftarrow (P_p) \text{ AND data}$   $p = 1-2$

```
Example:  ANDP2: ANL P2,#0F0H      ;'AND' PORT 2 CONTENTS
           ;WITH MASK 'F0' HEX
           ;(CLEAR P20-23)
```



## MCS-48 INSTRUCTION SET

### ANLD Pp,A Logical AND Port 4-7 with Accumulator Mask

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 0 | 0 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 1 | 1 | p | p |
|---|---|---|---|

 9CH-9FH

**Description:** This is a 2-cycle instruction. Data on port 'p' is logically ANDed with the digit mask contained in accumulator bits 0-3.

**Operation:**  $(Pp) \leftarrow (Pp) \text{ AND } (A0-3) \text{ } p = 4-7$

Note: The mapping of port 'p' to opcode bits 0-1 is as follows:

| 10 | Port |
|----|------|
| 00 | 4    |
| 01 | 5    |
| 10 | 6    |
| 11 | 7    |

**Example:** ANDP4: ANLD P4,A ;'AND' PORT 4 CONTENTS  
;WITH ACC BITS 0-3

### CALL address Subroutine Call

**Encoding:**

|     |    |    |   |
|-----|----|----|---|
| a10 | a9 | a8 | 1 |
|-----|----|----|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 0 |
|---|---|---|---|

|    |    |    |    |
|----|----|----|----|
| a7 | a6 | a5 | a4 |
|----|----|----|----|

|    |    |    |    |
|----|----|----|----|
| a3 | a2 | a1 | a0 |
|----|----|----|----|

| Page | Hex Op Code |
|------|-------------|
| 0    | 14          |
| 1    | 34          |
| 2    | 54          |
| 3    | 74          |
| 4    | 94          |
| 5    | B4          |
| 6    | D4          |
| 7    | F4          |

**Description:** This is a 2-cycle instruction. The program counter and PSW bits 4-7 are saved in the stack. The stack pointer (PSW bits 0-2) is updated. Program control is then passed to the location specified by 'address'. PC bit 11 is determined by the most recent SEL MB instruction.

A CALL cannot begin in locations 2046-2047 or 4094-4095. Execution continues at the instruction following the CALL upon return from the subroutine.

**Operation:**  $((SP)) \leftarrow (PC), (PSW \text{ 4-7})$   
 $(SP) \leftarrow (SP) + 1$   
 $(PC8-10) \leftarrow (addr8-10)$   
 $(PC0-7) \leftarrow addr0-7$   
 $(PC11) \leftarrow DBF$



## MCS-48 INSTRUCTION SET

---

**Example:** Add three groups of two numbers. Put subtotals in locations 50, 51 and total in location 52.

|                   |                           |
|-------------------|---------------------------|
| MOV R0,#50        | ;MOVE '50' DEC TO ADDRESS |
|                   | ;REG 0                    |
| BEGADD: MOV A,R1  | ;MOVE CONTENTS OF REG 1   |
|                   | ;TO ACC                   |
| ADD A,R2          | ;ADD REG 2 TO ACC         |
| CALL SUBTOT       | ;CALL SUBROUTINE 'SUBTOT' |
| ADDC A,R3         | ;ADD REG 3 TO ACC         |
| ADDC A,R4         | ;ADD REG 4 TO ACC         |
| CALL SUBTOT       | ;CALL SUBROUTINE 'SUBTOT' |
| ADDC A,R5         | ;ADD REG 5 TO ACC         |
| ADDC A,R6         | ;ADD REG 6 TO ACC         |
| CALL SUBTOT       | ;CALL SUBROUTINE 'SUBTOT' |
| SUBTOT: MOV @R0,A | ;MOVE CONTENTS OF ACC TO  |
|                   | ;LOCATION ADDRESSED BY    |
|                   | ;REG 0                    |
| INC R0            | ;INCREMENT REG 0          |
| RET               | ;RETURN TO MAIN PROGRAM   |

### CLR A Clear Accumulator

---

**Encoding:** 0010 0111 27H

**Description:** The contents of the accumulator are cleared to zero.

**Operation:**  $A \leftarrow 0$

### CLR C Clear Carry Bit

---

**Encoding:** 1001 0111 97H

**Description:** During normal program execution, the carry bit can be set to one by the ADD, ADDC, RLC, CPL C, RRC, and DAA instructions. This instruction resets the carry bit to zero.

**Operation:**  $C \leftarrow 0$

### CLR F1 Clear Flag 1 (Not in 8021H, 8022H)

---

**Encoding:** 1010 0101 A5H

**Description:** Flag 1 is cleared to zero.

**Operation:**  $(F1) \leftarrow 0$



## MCS-48 INSTRUCTION SET

### CLR F0 Clear Flag 0

(Not in 8021H, 8022H)

Encoding: 

|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|

 85H

Description: Flag 0 is cleared to zero.

Operation:  $(F0) \leftarrow 0$

### CPL A Complement Accumulator

Encoding: 

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|

 37H

Description: The contents of the accumulator are complemented. This is strictly a one's complement. Each one is changed to zero and vice-versa.

Operation:  $(A) \leftarrow \text{NOT } (A)$

Example: Assume accumulator contains 01101010.

CPLA: CPL A ;ACC CONTENTS ARE COMPLEMENTED TO 10010101

### CPL C Complement Carry Bit

Encoding: 

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|

 A7H

Description: The setting of the carry bit is complemented; one is changed to zero, and zero is changed to one.

Operation:  $(C) \leftarrow \text{NOT } (C)$

Example: Set C to one; current setting is unknown.

CTO1: CLR C ;C IS CLEARED TO ZERO  
CPL C ;C IS SET TO ONE

### CPL F0 Complement Flag 0

(Not in 8021H, 8022H)

Encoding: 

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|

 95H

Description: The setting of flag 0 is complemented; one is changed to zero, and zero is changed to one.

Operation:  $F0 \leftarrow \text{NOT } (F0)$

### CPL F1 Complement Flag 1

(Not in 8021H, 8022H)

Encoding: 

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|

 B5H

Description: The setting of flag 1 is complemented; one is changed to zero, and zero is changed to one.

Operation:  $(F1) \leftarrow \text{NOT } (F1)$



**DA A**    **Decimal Adjust Accumulator**

**Description:** The 8-bit accumulator value is adjusted to form two 4-bit Binary Coded Decimal (BCD) digits following the binary addition of BCD numbers. The carry bit C is affected. If the contents of bits 0-3 are greater than nine, or if AC is one, the accumulator is incremented by six.

The four high-order bits are then checked. If bits 4-7 exceed nine, or if C is one, these bits are increased by six. If an overflow occurs, C is set to one.

**Example:** Assume accumulator contains 10011011.

DA A ;ACC Adjusted to 00000001  
:WITH C SET

CAC 7 43 0

0 0 1 0 0 1 1 0 1 1

0000 0110

0 1 1 0 1 0 0 0 0 1

0 1 1 0

1 0 0 0 0 0 0 0 1

ADD SIX TO BITS 0-7

ADD SIX TO BITS 4-7

## OVERFLOW TO C

## DEC A Decrement Accumulator

**Description:** The contents of the accumulator are decremented by one. The carry flag is not affected.

**Operation:**  $(A) \leftarrow (A) - 1$

**Example:** Decrement contents of external data memory location 63.

MOV R0,#3FH

MOVX A, @R0

DEC A

MOVX @R0,A

```
;MOVE '3F' HEX TO REG 0
```

;MOVE CONTENTS OF LOCATION 63

:TO ACC

:DECREMENT ACC

```
;MOVE CONTENTS OF ACC TO
```

;LOCATION 63 IN EXPANDED

;MEMORY

## DEC Rr Decrement Register

(Not in 8021H, 8022H)

**Description:** The contents of working register 'r' are decremented by one.

**Operation:**  $(Rr) \leftarrow (Rr) - 1$

 $R = 0.7$ 

**Example:** DECR1: DEC R1

;DECREMENT CONTENTS OF REG 1



## MCS-48 INSTRUCTION SET

### DIS I Disable External Interrupt (Not in 8021H)

**Encoding:**

|      |      |
|------|------|
| 0001 | 0101 |
|------|------|

 15H

**Description:** External interrupts are disabled. A low signal on the interrupt input pin has no effect.

### DIS TCNTI Disable Timer/Counter Interrupt (Not in 8021H)

**Encoding:**

|      |      |
|------|------|
| 0011 | 0101 |
|------|------|

 35H

**Description:** Timer/counter interrupts are disabled. Any pending timer interrupt request is cleared. The interrupt sequence is not initiated by an overflow, but the timer flag is set and time accumulation continues.

### DJNZ Rr, address Decrement Register and Test

**Encoding:**

|      |      |
|------|------|
| 1110 | 1rrr |
|------|------|

|    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|
| a7 | a6 | a5 | a4 | a3 | a2 | a1 | a0 |
|----|----|----|----|----|----|----|----|

 E8H-EFH

**Description:** This is a 2-cycle instruction. Register 'r' is decremented, then tested for zero. If the register contains all zeros, program control falls through to the next instruction. If the register contents are not zero, control jumps to the specified 'address'.

The address in this case must evaluate to 8-bits, that is, the jump must be to a location within the current 256-location page.

**Operation:**  $(Rr) \leftarrow (Rr) - 1$   $r = 0-7$

If Rr not 0

$(PC0-7) \leftarrow \text{addr}$

**Note:** A 12-bit address specification does not cause an error if the DJNZ instruction and the jump target are on the same page. If the DJNZ instruction begins in location 255 of a page, it must jump to a target address on the following page.

**Example:** Increment values in data memory locations 50-54.

MOV R0,#50 ;MOVE '50' DEC TO ADDRESS

;REG 0

MOV R3,#5 ;MOVE '5' DEC TO COUNTER

;REG 3

INCRT: INC @R0 ;INCREMENT CONTENTS OF

;LOCATION ADDRESSED BY

;REG 0

INC R0 ;INCREMENT ADDRESS IN REG 0

DJNZ R3, INCRT ;DECREMENT REG 3 — JUMP TO

;'INCRT' IF REG 3 NONZERO

NEXT —

;'NEXT' ROUTINE EXECUTED

;IF R3 IS ZERO



## MCS-48 INSTRUCTION SET

### EN I Enable External Interrupt (Not in 8021H)

**Encoding:** 0000 0101 05H

**Description:** External interrupts are enabled. A low signal on the interrupt input pin initiates the interrupt sequence.

### EN TCNTI Enable Timer/Counter Interrupt (Not in 8021H)

**Encoding:** 0010 0101 25H

**Description:** Timer/counter interrupts are enabled. An overflow of the timer/counter initiates the interrupt sequence.

### ENT0 CLK Enable Clock Output (Not in 8021H, 8022H)

**Encoding:** 0111 0101 75H

**Description:** The test 0 pin is enabled to act as the clock output. This function is disabled by a system reset.

**Example:** EMTST0: ENT0 CLK ;ENABLE T0 AS CLOCK OUTPUT

### IN A,Pp Input Port or Data to Accumulator

**Encoding:** 0000 10pp 09H-0AH

**Description:** This is a 2-cycle instruction. Data present on port 'p' is transferred (read) to the accumulator. In the 8021 IN A,P2 inputs P20-P23 to A0-A3 while A4-A7 is set to zero.

**Operation:** (A) ← (Pp) P = 1-2

**Example:** INP12: IN A,P1 ;INPUT PORT 1 CONTENTS TO ACC  
                   MOV R6,A ;MOVE ACC CONTENTS TO REG 6  
                   IN A,P2 ;INPUT PORT 2 CONTENTS TO ACC  
                   MOV R7,A ;MOVE ACC CONTENTS TO REG 7

### INC A Increment Accumulator

**Encoding:** 0001 0111 17H

**Description:** The contents of the accumulator are incremented by one. Carry is not affected.

**Operation:** (A) ← (A) + 1



## MCS-48 INSTRUCTION SET

**Example:** Increment contents of location 100 in external data memory.

```
INCA: MOV R0,#100      ;MOVE '100' DEC TO ADDRESS REG 0
      MOVX A,@R0        ;MOVE CONTENTS OF LOCATION
                        ;100 TO ACC
      INC A             ;INCREMENT A
      MOVX @R0,A        ;MOVE ACC CONTENTS TO
                        ;LOCATION 101
```

### INC R<sub>r</sub> Increment Register

**Encoding:** 0001 1rrr 18H-1FH

**Description:** The contents of working register 'r' are incremented by one.

**Operation:**  $(Rr) \leftarrow (Rr) + 1$  R = 0-7

**Example:** INCR0: INC R0 ;INCREMENT CONTENTS OF REG 0

### INC @R<sub>r</sub> Increment Data Memory Location

**Encoding:** 0001 000r 10H-11H

**Description:** The contents of the resident data memory location addressed by register 'r' bits 0-5\* are incremented by one.

**Operation:**  $((Rr)) \leftarrow ((Rr)) + 1$  r = 0-1

**Example:** INCDM: MOV R1,#03FH ;MOVE ONES TO REG 1  
INC @R1 ;INCREMENT LOCATION 63

### IN A,P0 Input of Port 0 Data to Accumulator (8021H, 8022H Only)

Same as INS A, BUS except no  $\overline{RD}$  pulse generated. 80H

### INS A,BUS Strobed Input of BUS Data to Accumulator

**Encoding:** 0000 1000 08H

**Description:** This is a 2-cycle instruction. Data present on the BUS port is transferred (read) to the accumulator when the  $\overline{RD}$  pulse is dropped.  
(Refer to section on programming memory expansion for details).

**Operation:**  $(A) \leftarrow (BUS)$

**Example:** INPBUS: INS A,BUS ;INPUT BUS CONTENTS TO ACC



# MCS-48 INSTRUCTION SET

**JBb address    Jump If Accumulator Bit is Set**

(Not in 8021H, 8022H)

**Encoding:**

|    |    |    |   |
|----|----|----|---|
| b2 | b1 | b0 | 1 |
|----|----|----|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 1 | 0 |
|---|---|---|---|

|    |    |    |    |
|----|----|----|----|
| a7 | a6 | a5 | a4 |
|----|----|----|----|

|    |    |    |    |
|----|----|----|----|
| a3 | a2 | a1 | a0 |
|----|----|----|----|

| Accumulator Bit | Hex Op Code |
|-----------------|-------------|
| 0               | 12          |
| 1               | 32          |
| 2               | 52          |
| 3               | 72          |
| 4               | 92          |
| 5               | B2          |
| 6               | D2          |
| 7               | F2          |

**Description:** This is a 2-cycle instruction. Control passes to the specified address if accumulator bit 'b' is set to one.

**Operation:** b = 0-7

$$(PC_{0-7}) \leftarrow \text{addr}$$

If  $Bb = 1$

$$(PC) = (PC) + 2$$

If  $Bb = 0$

**Example:** JB4IS1: JB4 NEXT ;JUMP TO 'NEXT' ROUTINE

```

;IF ACC BIT 4 = 1

```

**JC address    Jump If Carry Is Set**

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | 0 |
|---|---|---|---|

|    |    |    |    |
|----|----|----|----|
| a7 | a6 | a5 | a4 |
|----|----|----|----|

|    |    |    |    |
|----|----|----|----|
| a3 | a2 | a1 | a0 |
|----|----|----|----|

 F6H

**Description:** This is a 2-cycle instruction. Control passes to the specified address if the carry bit is set to one.

**Operation:**  $(PC_{0-7}) \leftarrow \text{addr}$  If  $C = 1$

If  $C = 1$

$$(PC) = (PC) + 2$$

If  $C = 0$

**Example:** JC1: JC OVFLOW ;JUMP TO 'OVFLOW' ROUTINE

:IF C=1

**JF0 address    Jump If Flag 0 Is Set**

(Not in 8021H, 8022H)

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 0 | 1 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | 0 |
|---|---|---|---|

|    |    |    |    |
|----|----|----|----|
| a7 | a6 | a5 | a4 |
|----|----|----|----|

|    |    |    |    |
|----|----|----|----|
| a3 | a2 | a1 | a0 |
|----|----|----|----|

 B6H

**Description:** This is a 2-cycle instruction. Control passes to the specified address if flag 0 is set to one.

**Operation:**  $(PC_{0-7}) \leftarrow \text{addr}$  If  $F0 = 1$

If  $F0 = 1$

$$(PC) = (PC) + 2$$

If  $F_0 = 0$

**Example:** JF0IS1: JF0 TOTAL ;JUMP TO 'TOTAL' ROUTINE IF F0 = 1



# MCS-48 INSTRUCTION SET

**JF1 address**    **Jump If Flag 1 Is Set**  
(Not in 8021H, 8022H)

**Encoding:**    0 1 1 1   0 1 1 0            a7 a6 a5 a4   a3 a2 a1 a0            76H

**Description:** This is a 2-cycle instruction. Control passes to the specified address if flag 1 is set to one.

**Operation:**  $(PC_{0-7}) \leftarrow \text{addr}$  If  $F1 = 1$   
 $(PC) = (PC + 2)$  If  $F1 = 0$

**Example:** JF1IS1: JF1 FILBUF ;JUMP TO 'FILBUF'  
:ROUTINE IF F1 = 1

**JMP address**    **Direct Jump within 2K Block** ✓

**Encoding:**

|                 |                |                |   |   |   |   |   |
|-----------------|----------------|----------------|---|---|---|---|---|
| a <sub>10</sub> | a <sub>9</sub> | a <sub>8</sub> | 0 | 0 | 1 | 0 | 0 |
|-----------------|----------------|----------------|---|---|---|---|---|

|                |                |                |                |                |                |                |                |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| a <sub>7</sub> | a <sub>6</sub> | a <sub>5</sub> | a <sub>4</sub> | a <sub>3</sub> | a <sub>2</sub> | a <sub>1</sub> | a <sub>0</sub> |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|

| Page | Hex Op Code |
|------|-------------|
| 0    | 04          |
| 1    | 24          |
| 2    | 44          |
| 3    | 64          |
| 4    | 84          |
| 5    | A4          |
| 6    | C4          |
| 7    | E4          |

**Description:** This is a 2-cycle instruction. Bits 0-10 of the program counter are replaced with the directly-specified address. The setting of PC bit 11 is determined by the most recent SELECT MB instruction.

**Operation:** (PC8-10) ← addr 8-10  
(PC0-7) ← addr 0-7  
(PC11) ← DBF

|                 |            |                              |
|-----------------|------------|------------------------------|
| <b>Example:</b> | JMP SUBTOT | ;JUMP TO SUBROUTINE 'SUBTOT' |
|                 | JMP \$-6   | ;JUMP TO INSTRUCTION SIX     |
|                 |            | ;LOCATIONS BEFORE CURRENT    |
|                 |            | ;LOCATION                    |
|                 | JMP 2FH    | ;JUMP TO ADDRESS '2F' HEX    |

### JMPP @A Indirect Jump within Page

**Encoding:**

|      |      |
|------|------|
| 1011 | 0011 |
|------|------|

 B3H

**Description:** This is a 2-cycle instruction. The contents of the program memory location pointed to by the accumulator are substituted for the 'page' portion of the program counter (PC bits 0-7).



## MCS-48 INSTRUCTION SET

**Operation:**  $(PC_{0-7}) \leftarrow ((A))$

**Example:** Assume accumulator contains 0FH.

JMPPAG: JMPP @A ;JUMP TO ADDRESS STORED IN  
;LOCATION 15 IN CURRENT PAGE

### JNC address Jump If Carry Is Not Set

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 1 | 1 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | 0 |
|---|---|---|---|

|    |    |    |    |
|----|----|----|----|
| a7 | a6 | a5 | a4 |
|----|----|----|----|

|    |    |    |    |
|----|----|----|----|
| a3 | a2 | a1 | a0 |
|----|----|----|----|

 E6H

**Description:** This is a 2-cycle instruction. Control passes to the specified address if the carry bit is not set, that is, equals zero.

**Operation:**  $(PC_{0-7}) \leftarrow \text{addr}$  If C = 0  
 $(PC) = (PC) + 2$  If C = 1

**Example:** JC0: JNC NOVFO ;JUMP TO 'NOVFO' ROUTINE  
;IF C = 0

### JNI address Jump If Interrupt Input is Low (Not in 8021H, 8022H)

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | 0 |
|---|---|---|---|

|    |    |    |    |
|----|----|----|----|
| a7 | a6 | a5 | a4 |
|----|----|----|----|

|    |    |    |    |
|----|----|----|----|
| a3 | a2 | a1 | a0 |
|----|----|----|----|

 86H

**Description:** This is a 2-cycle instruction. Control passes to the specified address if the interrupt input signal is low (= 0), that is, an external interrupt has been signaled. (This signal initiates an interrupt service sequence if the external interrupt is enabled.)

**Operation:**  $(PC_{0-7}) \leftarrow \text{addr}$  If I = 0  
 $(PC) = (PC) + 2$  If I = 1

**Example:** LOC 3: JNI EXTINT ;JUMP TO 'EXTINT' ROUTINE  
;IF I = 0

### JNT0 address Jump If Test 0 Is Low (Not in 8021H)

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 1 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | 0 |
|---|---|---|---|

|    |    |    |    |
|----|----|----|----|
| a7 | a6 | a5 | a4 |
|----|----|----|----|

|    |    |    |    |
|----|----|----|----|
| a3 | a2 | a1 | a0 |
|----|----|----|----|

 26H

**Description:** This is a 2-cycle instruction. Control passes to the specified address, if the test 0 signal is low.

**Operation:**  $(PC_{0-7}) \leftarrow \text{addr}$  If T0 = 0  
 $(PC) = (PC) + 2$  If T0 = 1

**Example:** JT0LOW: JNT0 60 ;JUMP TO LOCATION 60 DEC  
;IF T0 = 0



2830

# MCS-48 INSTRUCTION SET

## JNT1 address Jump If Test 1 Is Low

Encoding: 0100 0110 a7 a6 a5 a4 a3 a2 a1 a0 46H

Description: This is a 2-cycle instruction. Control passes to the specified address, if the test 1 signal is low.

Operation: (PC0-7) ← addr If T1 = 0  
(PC) = (PC) + 2 If T1 = 1

## JNZ address Jump If Accumulator Is Not Zero

Encoding: 1001 0110 a7 a6 a5 a4 a3 a2 a1 a0 96H

Description: This is a 2-cycle instruction. Control passes to the specified address if the accumulator contents are nonzero at the time this instruction is executed.

Operation: (PC0-7) ← addr If A ≠ 0  
(PC) = (PC) + 2 If A = 0

Example: JACCNO: JNZ 0ABH ;JUMP TO LOCATION 'AB' HEX  
;IF ACC VALUE IS NONZERO

## JTF address Jump If Timer Flag Is Set

Encoding: 0001 0110 a7 a6 a5 a4 a3 a2 a1 a0 16H

Description: This is a 2-cycle instruction. Control passes to the specified address if the timer flag is set to one, that is, the timer/counter register has overflowed. Testing the timer flag resets it to zero. (This overflow initiates an interrupt service sequence if the timer-overflow interrupt is enabled.)

Operation: (PC0-7) ← addr If TF = 1  
(PC) = (PC) + 2 If TF = 0

Example: JTF1: JTF TIMER ;JUMP TO 'TIMER' ROUTINE  
;IF TF = 1

## JT0 address Jump If Test 0 Is High (Not in 8021H)

Encoding: 0011 0110 a7 a6 a5 a4 a3 a2 a1 a0 36H

Description: This is a 2-cycle instruction. Control passes to the specified address if the test 0 signal is high (= 1).

Operation: (PC0-7) ← addr If T0 = 1  
(PC) = (PC) + 2 If T0 = 0

Example: JT0HI: JT0 53 ;JUMP TO LOCATION 53 DEC  
;IF T0 = 1



## MCS-48 INSTRUCTION SET

### JT1 address    Jump If Test 1 Is High

**Encoding:**    0 1 0 1   0 1 1 0    a7 a6 a5 a4   a3 a2 a1 a0    56H

**Description:** This is a 2-cycle instruction. Control passes to the specified address if the test 1 signal is high (= 1).

**Operation:**    (PC0-7) ← addr                      If T1 = 1  
                      (PC) = (PC) + 2                      If T1 = 0

**Example:**    JT1HI: JT1 COUNT                      ;JUMP TO 'COUNT' ROUTINE  
    ;IF T1 = 1

### JZ address    Jump If Accumulator Is Zero

**Encoding:**    1 1 0 0   0 1 1 0    a7 a6 a5 a4   a3 a2 a1 a0    C6H

**Description:** This is a 2-cycle instruction. Control passes to the specified address if the accumulator contains all zeros at the time this instruction is executed.

**Operation:**    (PC0-7) ← addr                      If A = 0  
                      (PC) = (PC) + 2                      If A ≠ 1

**Example:**    JACCO: JZ 0A3H                      ;JUMP TO LOCATION 'A3' HEX  
    ;IF ACC VALUE IS ZERO

### MOV A,#data    Move Immediate Data to Accumulator

**Encoding:**    0 0 1 0   0 0 1 1    d7 d6 d5 d4   d3 d2 d1 d0    23H

**Description:** This is a 2-cycle instruction. The 8-bit value specified by 'data' is loaded in the accumulator.

**Operation:**    (A) ← data

**Example:**    MOV A,#0A3H                      ;MOVE 'A3' HEX TO ACC

### MOV A,PSW    Move PSW Contents to Accumulator                      (Not in 8021H, 8022H)

**Encoding:**    1 1 0 0   0 1 1 1    C7H

**Description:** The contents of the program status word are moved to the accumulator.

**Operation:**    (A) ← (PSW)

**Example:**    Jump to 'RB1SET' routine if PSW bank switch, bit 4, is set.  
                      BSCHK: MOV A,PSW                      ;MOVE PSW CONTENTS TO ACC  
    JB4 RB1SET                      ;JUMP TO 'RB1SET' IF ACC BIT 4 = 1



## MCS-48 INSTRUCTION SET

### MOV A,Rr Move Register Contents to Accumulator

**Encoding:** 11111rrr F8H-FFH

**Description:** 8-bits of data are moved from working register 'r' into the accumulator.

**Operation:**  $(A) \leftarrow (Rr)$   $r = 0-7$

**Example:** MAR: MOV A,R3 ;MOVE CONTENTS OF REG 3 TO ACC

### MOV A,@Rr Move Data Memory Contents to Accumulator

**Encoding:** 1111000r F0H-F1H

**Description:** The contents of the resident data memory location addressed by bits 0-5\* of register 'r' are moved to the accumulator. Register 'r' contents are unaffected.

**Operation:**  $(A) \leftarrow ((Rr))$   $r = 0-1$

**Example:** Assume R1 contains 00110110.

MADM: MOV A,@R1 ;MOVE CONTENTS OF DATA MEM  
;LOCATION 54 TO ACC

### MOV A,T Move Timer/Counter Contents to Accumulator

**Encoding:** 01000010 42H

**Description:** The contents of the timer/event-counter register are moved to the accumulator.

**Operation:**  $(A) \leftarrow (T)$

**Example:** Jump to "EXIT" routine when timer reaches '64', that is, when bit 6 set — assuming initialization 64,

TIMCHK: MOV A,T ;MOVE TIMER CONTENTS TO ACC  
JB6 EXIT ;JUMP TO 'EXIT' IF ACC BIT 6 = 1

### MOV PSW,A Move Accumulator Contents to PSW

(Not in 8021H, 8022H)

**Encoding:** 11010111 D7H

**Description:** The contents of the accumulator are moved into the program status word. All condition bits and the stack pointer are affected by this move.

**Operation:**  $(PSW) \leftarrow (A)$

**Example:** Move up stack pointer by two memory locations, that is, increment the pointer by one.

INCPTR: MOV A,PSW ;MOVE PSW CONTENTS TO ACC  
INC A ;INCREMENT ACC BY ONE  
MOV PSW,A ;MOVE ACC CONTENTS TO PSW



# MCS-48 INSTRUCTION SET

### MOV R<sub>r</sub>,A    Move Accumulator Contents to Register

**Encoding:**

|         |         |
|---------|---------|
| 1 0 1 0 | 1 r r r |
|---------|---------|

 A8H-AFH

**Description:** The contents of the accumulator are moved to register 'r'.

**Operation:**  $(Rr) \leftarrow (A)$   $r = 0.7$

**Example:** MRA: MOV R0,A ;MOVE CONTENTS OF ACC TO REG 0

### MOV Rr,#data    Move Immediate Data to Register

**Encoding:**

|         |  |
|---------|--|
| 1 0 1 1 | 1 r <sub>2</sub> r <sub>1</sub> r <sub>0</sub> |
|---------|--|

|   |   |
|---|---|
| d <sub>7</sub> d <sub>6</sub> d <sub>5</sub> d <sub>4</sub> | d <sub>3</sub> d <sub>2</sub> d <sub>1</sub> d <sub>0</sub> |
|---|---|

 B8H-BFH

**Description:** This is a 2-cycle instruction. The 8-bit value specified by 'data' is moved to register 'r'.

**Operation:**  $(Rr) \leftarrow \text{data}$   $r = 0-7$

**Examples:** MIR4: MOV R4,#HEXTEN ;THE VALUE OF THE SYMBOL 'HEXTEN'  
;IS MOVED INTO REG 4

MIR 5: MOV R5,#PI\*(R\*R) ;THE VALUE OF THE EXPRESSION

```

;PI*(R*R) IS MOVED INTO REG 5

```

MIR 6: MOV R6, #0ADH           : 'AD' HEX IS MOVED INTO REG 6

### MOV @Rr,A Move Accumulator Contents to Data Memory

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 0 | 1 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 0 | r |
|---|---|---|---|

    A0H-A1H

**Description:** The contents of the accumulator are moved to the resident data memory location whose address is specified by bits 0-5\* of register 'r'. Register 'r' contents are unaffected.

**Operation:**  $((Rr)) \leftarrow (A)$   $r = 0-1$

**Example:** Assume R0 contains 00000111.

```
MDMA: MOV @R0,A      ;MOVE CONTENTS OF ACC TO
                     ;LOCATION 7 (REG 7)
```

### MOV @Rr,#data    Move Immediate Data to Data Memory

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 0 | 1 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 0 | r |
|---|---|---|---|

|    |    |    |    |
|----|----|----|----|
| d7 | d6 | d5 | d4 |
|----|----|----|----|

|    |    |    |    |
|----|----|----|----|
| d3 | d2 | d1 | d0 |
|----|----|----|----|

 B0H-B1H

**Description:** This is a 2-cycle instruction. The 8-bit value specified by 'data' is moved to the resident data memory location addressed by register 'r', bits 0-5\*.

**Operation:**  $((Rr)) \leftarrow \text{data}$   $r = 0-1$

**Examples:** Move the hexadecimal value AC3F to locations 62-63.

MIDM: MOV R0,#62 :MOVE '62' DEC TO ADDR REG 0

MOV @R0,#0ACH :MOVE 'AC' HEX TO LOCATION 62

```
INC R0 ;INCREMENT REG 0 TO '63'
```

```
MOV @R0,#3FH      ;MOVE '3F' HEX TO LOCATION 63
```



## MCS-48 INSTRUCTION SET

### MOV T,A    Move Accumulator Contents to Timer/Counter

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 1 | 0 |
|---|---|---|---|

    62H

**Description:** The contents of the accumulator are moved to the timer/event-counter register.

**Operation:** (T) ← (A)

**Example:** Initialize and start event counter.

```

INITEC: CLR A           ;CLEAR ACC TO ZEROS
         MOV T,A         ;MOVE ZEROS TO EVENT COUNTER
         STRT CNT        ;START COUNTER
    
```

### MOVD A,Pp    Move Port 4-7 Data to Accumulator

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 1 | 1 | p | p |
|---|---|---|---|

    0CH-0FH

**Description:** This is a 2-cycle instruction. Data on 8243 port 'p' is moved (read) to accumulator bits 0-3. Accumulator bits 4-7 are zeroed.

**Operation:** (0-3) ← (Pp)                      p = 4-7  
 (4-7) ← 0

**Note:** Bits 0-7 of the opcode are used to represent ports 4-7. If you are coding in binary rather than assembly language, the mapping is as follows:

| Bits 1 0 | Port |
|----------|------|
| 0 0      | 4    |
| 0 1      | 5    |
| 1 0      | 6    |
| 1 1      | 7    |

**Example:** INPPT5: MOVD A,P5    ;MOVE PORT 5 DATA TO ACC  
    ;BITS 0-3, ZERO ACC BITS 4-7

### MOVD Pp,A    Move Accumulator Data to Port 4-7

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 1 | 1 | p | p |
|---|---|---|---|

    3CH-3FH

**Description:** This is a 2-cycle instruction. Data in accumulator bits 0-3 is moved (written) to 8243 port 'p'. Accumulator bits 4-7 are unaffected. (See NOTE above regarding port mapping.)

**Operation:** (Pp) ← (A0-3)                      p = 4-7

**Example:** Move data in accumulator to ports 4 and 5.

```

OUTP45: MOVD P4,A       ;MOVE ACC BITS 0-3 TO PORT 4
         SWAP A          ;EXCHANGE ACC BITS 0-3 AND 4-7
         MOVD P5,A       ;MOVE ACC BITS 0-3 TO PORT 5
    
```



## MCS-48 INSTRUCTION SET

### MOVP A,@A    Move Current Page Data to Accumulator

**Encoding:**

|      |      |
|------|------|
| 1010 | 0011 |
|------|------|

    A3H

**Description:** The contents of the program memory location addressed by the accumulator are moved to the accumulator. Only bits 0-7 of the program counter are affected, limiting the program memory reference to the current page. The program counter is restored *following* this operation.

**Operation:** (PC0-7) ← (A)

(A) ← ((PC))

Note: This is a 1-byte, 2-cycle instruction. If it appears in location 255 of a program memory page, @A addresses a location in the *following* page.

**Example:**    MOV128: MOV A,#128                    ;MOVE '128' DEC TO ACC  
                       MOV A,@A                    ;CONTENTS OF 129th LOCATION IN  
   ;CURRENT PAGE ARE MOVED TO ACC

### MOVP3 A,@A    Move Page 3 Data to Accumulator                       (Not in 8021H, 8022H)

**Encoding:**

|      |      |
|------|------|
| 1110 | 0011 |
|------|------|

    E3H

**Description:** This is a 2-cycle instruction. The contents of the program memory location (within page 3) addressed by the accumulator are moved to the accumulator. The program counter is restored following this operation.

**Operation:** (PC0-7) ← (A)

(PC8-11) ← 0011

(A) ← ((PC))

**Example:** Look up ASCII equivalent of hexadecimal code in table contained at the beginning of page 3. Note that ASCII characters are designated by a 7-bit code; the eighth bit is always reset.

TABSCH: MOV A,#0B8H                    ;MOVE 'B8' HEX TO ACC (10111000)  
                       ANL A,#7FH                    ;LOGICAL AND ACC TO MASK BIT  
   ;7 (00111000)  
                       MOVP3 A,@A                    ;MOVE CONTENTS OF LOCATION '38'  
   ;HEX IN PAGE 3 TO ACC (ASCII '8')

Access contents of location in page 3 labelled TAB1.

Assume current program location is not in page 3.

TABSCH: MOV A,#LOW TAB 1                ;ISOLATE BITS 0-7 OF LABEL  
   ;ADDRESS VALUE  
                       MOVP3 A,@A                    ;MOVE CONTENTS OF PAGE 3  
   ;LOCATION LABELED 'TAB1' TO ACC



# MCS-48 INSTRUCTION SET

## MOVX A,@Rr Move External-Data-Memory Contents to Accumulator

(Not in 8021H, 8022H)

|           |         |         |         |
|-----------|---------|---------|---------|
| Encoding: | 1 0 0 0 | 0 0 0 r | 80H-81H |
|-----------|---------|---------|---------|

**Description:** This is a 2-cycle instruction. The contents of the external data memory location addressed by register 'r' are moved to the accumulator. Register 'r' contents are unaffected. A read pulse is generated.

**Operation:**  $(A) \leftarrow ((Rr))$   $r = 0-1$

**Example:** Assume R1 contains 01110110.

MAXDM: MOVX A,@R1 ;MOVE CONTENTS OF LOCATION  
32 ;118 TO ACC

### MOVX @Rr,A Move Accumulator Contents to External Data Memory

(Not in 8021H, 8022H)

**Encoding:**

|         |         |
|---------|---------|
| 1 0 0 1 | 0 0 0 r |
|---------|---------|

 90H-91H

**Description:** This is a 2-cycle instruction. The contents of the accumulator are moved to the external data memory location addressed by register 'r'. Register 'r' contents are unaffected. A write pulse is generated.

**Operation:**  $((Rr)) \leftarrow A$   $r = 0-1$

**Example:** Assume R0 contains 11000111.

```
MXDMA: MOVX @R0,A      ;MOVE CONTENTS OF ACC TO
                        ;LOCATION 199 IN EXPANDED
                        ;DATA MEMORY
```

## NOP The NOP Instruction

**Encoding:**

|      |      |
|------|------|
| 0000 | 0000 |
|------|------|

 00H

**Description:** No operation is performed. Execution continues with the following instruction.

### ORL A,R<sub>r</sub> Logical OR Accumulator With Register Mask

**Encoding:**    0 1 0 0    1 r r r    48H-4FH

**Description:** Data in the accumulator is logically ORed with the mask contained in working register 'r'.

**Operation:**  $(A) \leftarrow (A) \text{ OR } (Rr)$   $r = 0-7$

**Example:** ORREG: ORL A,R4 ;'OR' ACC CONTENTS WITH  
:MASK IN REG 4



## MCS-48 INSTRUCTION SET

### ORL A,@Rr    Logical OR Accumulator With Memory Mask

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 0 | r |
|---|---|---|---|

    40H-41H

**Description:** Data in the accumulator is logically ORed with the mask contained in the resident data memory location referenced by register 'r', bits 0-5\*.

**Operation:** (A) ← (A) OR ((Rr))                      r = 0-1

**Example:**    ORDM: MOV R0,#3FH                      ;MOVE '3F' HEX TO REG 0  
                      ORL A,@R0                      ;'OR' ACC CONTENTS WITH MASK  
    ;IN LOCATION 63

### ORL A,#data    Logical OR Accumulator With Immediate Mask

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
|---|---|---|---|

|    |    |    |    |
|----|----|----|----|
| d7 | d6 | d5 | d4 |
|----|----|----|----|

|    |    |    |    |
|----|----|----|----|
| d3 | d2 | d1 | d0 |
|----|----|----|----|

    43H

**Description:** This is a 2-cycle instruction. Data in the accumulator is logically ORed with an immediately-specified mask.

**Operation:** (A) ← (A) OR data

**Example:**    ORID: ORL A,#'X'                      ;'OR' ACC CONTENTS WITH MASK  
    ;01011000 (ASCII VALUE OF 'X')

### ORL BUS,#data    Logical OR BUS With Immediate Mask    (Not in 8021H, 8022H)

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
|---|---|---|---|

|    |    |    |    |
|----|----|----|----|
| d7 | d6 | d5 | d4 |
|----|----|----|----|

|    |    |    |    |
|----|----|----|----|
| d3 | d2 | d1 | d0 |
|----|----|----|----|

    88H

**Description:** This is a 2-cycle instruction. Data on the BUS port is logically ORed with an immediately-specified mask. This instruction assumes prior specification of an 'OUTL BUS,A' instruction.

**Operation:** (BUS) ← (BUS) OR data

**Example:**    ORBUS: ORL BUS,#HEXMSK                      ;'OR' BUS CONTENTS WITH MASK  
    ;EQUAL VALUE OF SYMBOL 'HEXMSK'

### ORL Pp, #data    Logical OR Port 1 or 2 With Immediate Mask    (Not in 8021H, 8022H)

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 1 | 0 | p | p |
|---|---|---|---|

|    |    |    |    |
|----|----|----|----|
| d7 | d6 | d5 | d4 |
|----|----|----|----|

|    |    |    |    |
|----|----|----|----|
| d3 | d2 | d1 | d0 |
|----|----|----|----|

    89H-8AH

**Description:** This is a 2-cycle instruction. Data on port 'p' is logically ORed with an immediately-specified mask.

**Operation:** (Pp) ← (Pp) OR data                      p = 1-2

**Example:**    ORP1: ORL P1, #0FFH                      ;'OR' PORT 1 CONTENTS WITH MASK  
    ;'FF' HEX (SET PORT 1 TO ALL ONES)



## MCS-48 INSTRUCTION SET

### ORLD Pp,A    Logical OR Port 4-7 With Accumulator Mask

**Encoding:**

|         |         |
|---------|---------|
| 1 0 0 0 | 1 1 p p |
|---------|---------|

    8CH-8FH

**Description:** This is a 2-cycle instruction. Data on port 'p' is logically ORed with the digit mask contained in accumulator bits 0-3.

**Operation:**  $(Pp) \leftarrow (Pp) \text{ OR } (A0-3)$      $p = 4-7$

**Example:** ORP7: ORLD P7,A    ;'OR' PORT 7 CONTENTS WITH ACC  
;BITS 0-3

### OUTL P0,A    Output Accumulator Data to Port 0 (8021H, 8022H Only)

**Encoding:**

|         |         |
|---------|---------|
| 1 0 0 1 | 0 0 0 0 |
|---------|---------|

    90H

### OUTL BUS,A    Output Accumulator Data to BUS (Not in 8021H, 8022H)

**Encoding:**

|         |         |
|---------|---------|
| 0 0 0 0 | 0 0 1 0 |
|---------|---------|

    02H

**Description:** This is a 2-cycle instruction. Data residing in the accumulator is transferred (written) to the BUS port and latched. The latched data remains valid until altered by another OUTL instruction. Any other instruction requiring use of the BUS port (except INS) destroys the contents of the BUS latch. This includes expanded memory operations (such as the MOVX instruction). Logical operations on BUS data (AND, OR) assume the OUTL BUS,A instruction has been issued previously.

Does not  
apply for  
OUTL  
P0,A  
OF 8021H,  
8022H

**Operation:**  $(BUS) \leftarrow (A)$

**Example:** OUTLBP: OUTL BUS,A    ;OUTPUT ACC CONTENTS TO BUS

### OUTL Pp,A    Output Accumulator Data to Port 1 or 2

**Encoding:**

|         |         |
|---------|---------|
| 0 0 1 1 | 1 0 p p |
|---------|---------|

    39H-3AH

**Description:** This is a 2-cycle instruction. Data residing in the accumulator is transferred (written) to port 'p' and latched.

**Operation:**  $(Pp) \leftarrow (A)$      $p = 1-2$

**Example:** OUTLP: MOV A,R7    ;MOVE REG 7 CONTENTS TO ACC  
              OUTL P2,A    ;OUTPUT ACC CONTENTS TO PORT 2  
              MOV A,R6    MOVE REG 6 CONTENTS TO ACC  
              OUTL P1,A    ;OUTPUT ACC CONTENTS TO PORT 1



## MCS-48 INSTRUCTION SET

### RAD Move Conversion Result Register to Accumulator (8022H Only)

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
|---|---|---|---|

 80H

**Description:** This is a 2-cycle instruction. The contents of the A/D conversion result register are moved to the accumulator.

**Operation:** (A) ← (CRR)

### RET Return Without PSW Restore

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
|---|---|---|---|

 83H

**Description:** This is a 2-cycle instruction. The stack pointer (PSW bits 0-2) is decremented. The program counter is then restored from the stack. PSW bits 4-7 are not restored.

**Operation:** (SP) ← (SP)-1  
(PC) ← ((SP))

### RETI Return From Interrupt (8022H Only)

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 0 | 0 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
|---|---|---|---|

 93H

**Description:** This is a 2-cycle instruction. The stack pointer is decremented and the program counter is restored from the stack. Interrupt input logic is re-enabled.

**Operation:** (SP) ← (SP)-1  
(PC) ← ((SP)) + 1

### RETR Return with PSW Restore (Not in 8021H, 8022H)

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 0 | 0 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
|---|---|---|---|

 93H

**Description:** This is a 2-cycle instruction. The stack pointer is decremented. The program counter and bits 4-7 of the PSW are then restored from the stack. Note that RETR should be used to return from an interrupt, but should not be used within the interrupt service routine as it signals the end of an interrupt routine by resetting the Interrupt in Progress flipflop.

**Operation:** (SP) ← (SP)-1  
(PC) ← ((SP))  
(PSW 4-7) ← ((SP))



## MCS-48 INSTRUCTION SET

### RL A Rotate Left without Carry

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 1 | 1 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | 1 |
|---|---|---|---|

 E7H

**Description:** The contents of the accumulator are rotated left one bit. Bit 7 is rotated into the bit 0 position.

**Operation:**  $(A_n + 1) \leftarrow (A_n)$   
 $(A_0) \leftarrow (A_7)$   $n = 0-6$

**Example:** Assume accumulator contains 10110001.  
 RLNC: RL A ;NEW ACC CONTENTS ARE 01100011

### RLC A Rotate Left through Carry

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | 1 |
|---|---|---|---|

 F7H

**Description:** The contents of the accumulator are rotated left one bit. Bit 7 replaces the carry bit; the carry bit is rotated into the bit 0 position.

**Operation:**  $(A_n + 1) \leftarrow (A_n)$   $n = 0-6$   
 $(A_0) \leftarrow (C)$   
 $(C) \leftarrow (A_7)$

**Example:** Assume accumulator contains a 'signed' number; isolate sign without changing value.

|             |                             |
|-------------|-----------------------------|
| RLTC: CLR C | ;CLEAR CARRY TO ZERO        |
| RLC A       | ;ROTATE ACC LEFT, SIGN      |
|             | ;BIT (7) IS PLACED IN CARRY |
| RR A        | ;ROTATE ACC RIGHT — VALUE   |
|             | ;BITS 0-6 IS RESTORED,      |
|             | ;CARRY UNCHANGED, BIT 7     |
|             | ;IS ZERO                    |

### RR A Rotate Right without Carry

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | 1 |
|---|---|---|---|

 77H

**Description:** The contents of the accumulator are rotated right one bit. Bit 0 is rotated into the bit 7 position

**Operation:**  $(A_n) \leftarrow (A_n + 1)$   $n = 0-6$   
 $(A_7) \leftarrow (A_0)$

**Example:** Assume accumulator contains 10110001.  
 RRNC: RR A ;NEW ACC CONTENTS ARE 11011000



## MCS-48 INSTRUCTION SET

### RRC A Rotate Right through Carry

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | 1 |
|---|---|---|---|

 67H

**Description:** The contents of the accumulator are rotated right one bit. Bit 0 replaces the carry bit; the carry bit is rotated into the bit 7 position.

**Operation:**  $(A_n) \leftarrow (A_n + 1)$   $n = 0-6$   
 $(A_7) \leftarrow (C)$   
 $(C) \leftarrow (A_0)$

**Example:** Assume carry is not set and accumulator contains 10110001.  
 RRTC: RRC A ;CARRY IS SET AND ACC  
 ;CONTAINS 01011000

### SEL AN0 Select Analog Input Zero (8022H Only)

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 0 | 0 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 1 |
|---|---|---|---|

 95H

### SEL AN1 Select Analog Input One (8022H Only)

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 1 |
|---|---|---|---|

 85H

**Description:** One of the two analog inputs to the A/D converter is selected. The conversion process is started. Restarting a sequence deletes the sequence in progress.

### SEL MB0 Select Memory Bank 0 (Not in 8021H, 8022H)

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 1 | 1 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 1 |
|---|---|---|---|

 E5H

**Description:** PC bit 11 is set to zero on next JMP or CALL instruction. All references to program memory addresses fall within the range 0-2047.

**Operation:**  $(DBF) \leftarrow 0$

**Example:** Assume program counter contains 834 Hex.  
 SEL MB0 ;SELECT MEMORY BANK 0  
 JMP \$ + 20 ;JUMP TO LOCATION 58 HEX

### SEL MB1 Select Memory Bank 1 (Not in 8021H, 8022H)

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 1 |
|---|---|---|---|

 F5H

**Description:** PC bit 11 is set to one on next JMP or CALL instruction. All references to program memory addresses fall within the range 2048-4095.

**Operation:**  $(DBF) \leftarrow 1$



## MCS-48 INSTRUCTION SET

### SEL RB0 Select Register Bank 0 (Not in 8021H, 8022H)

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 |

 C5H

**Description:** PSW bit 4 is set to zero. References to working registers 0-7 address data memory locations 0-7. This is the recommended setting for normal program execution.

**Operation:** (BS) ← 0

### SEL RB1 Select Register Bank 1 (Not in 8021H, 8022H)

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 |

 D5H

**Description:** PSW bit 4 is set to one. References to working registers 0-7 address data memory locations 24-31. This is the recommended setting for interrupt service routines, since locations 0-7 are left intact. The setting of PSW bit 4 in effect at the time of an interrupt is restored by the RETR instruction when the interrupt service routine is completed.

**Operation:** (BS) ← 1

**Example:** Assume an external interrupt has occurred, control has passed to program memory location 3, and PSW bit 4 was zero before the interrupt.

**Operation:** LOC3: JN1 INIT ; JUMP TO ROUTINE 'INIT' IF  
INIT: MOV R7,A ; INTERRUPT INPUT IS ZERO  
SEL RB1 ; MOVE ACC CONTENTS TO  
MOV R7,#0FAH ; LOCATION 7  
; SELECT REG BANK 1  
; MOVE 'FA' HEX TO LOCATION 31

SEL RB0 ; SELECT REG BANK 0  
MOV A,R7 ; RESTORE ACC FROM LOCATION 7  
RETR ; RETURN — RESTORE PC AND PSW

### STOP TCNT Stop Timer/Event-Counter

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 |

 65H

**Description:** This instruction is used to stop both time accumulation and event counting.



## MCS-48 INSTRUCTION SET

**Example:** Disable interrupt, but jump to interrupt routine after eight overflows and stop timer. Count overflows in register 7.

|  |  |
|--|--|
| <pre> START: DIS TCNTI       CLR A       MOV T,A       MOV R7,A       STRT T MAIN: JTF COUNT       JMP MAIN COUNT: INC R7       MOV A,R7       JB3 INT       JMP MAIN       .       .       . INT: STOP TCNT       JMP 7H           </pre> | <pre> ;DISABLE TIMER INTERRUPT ;CLEAR ACC TO ZEROS ;MOVE ZEROS TO TIMER ;MOVE ZEROS TO REG 7 ;START TIMER ;JUMP TO ROUTINE 'COUNT' ;IF TF = 1 AND CLEAR TIMER FLAG ;CLOSE LOOP ;INCREMENT REG 7 ;MOVE REG 7 CONTENTS TO ACC ;JUMP TO ROUTINE 'INT' IF ACC ;BIT 3 IS SET (REG 7 = 8) ;OTHERWISE RETURN TO ROUTINE ;MAIN       .       .       . ;STOP TIMER ;JUMP TO LOCATION 7 (TIMER) ;INTERRUPT ROUTINE           </pre> |
|--|--|

### STRT CNT    Start Event Counter

**Encoding:**    0100 0101    45H

**Description:** The test 1 (T1) pin is enabled as the event-counter input and the counter is started. The event-counter register is incremented with each high-to-low transition on the T1 pin.

**Example:** Initialize and start event counter. Assume overflow is desired with first T1 input.

|  |   |
|--|---|
| <pre> STARTC: EN TCNTI       MOV A,#0FFH       MOV T,A       STRT CNT           </pre> | <pre> ;ENABLE COUNTER INTERRUPT ;MOVE 'FF' HEX (ONES) TO ACC ;MOVES ONES TO COUNTER ;ENABLE T1 AS COUNTER ;INPUT AND START           </pre> |
|--|---|



## MCS-48 INSTRUCTION SET

### STRT T Start Timer

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 1 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 1 |
|---|---|---|---|

 55H

**Description:** Timer accumulation is initiated in the timer register. The register is incremented every 32 instruction cycles. The prescaler which counts the 32 cycles is cleared but the timer register is not.

**Example:** Initialize and start timer.

```
STARTT: CLR A           ;CLEAR ACC TO ZEROS
          MOV T,A         ;MOVE ZEROS TO TIMER
          EN TCNTI        ;ENABLE TIMER INTERRUPT
          STRT T          ;START TIMER
```

### SWAP A Swap Nibbles within Accumulator

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 0 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 0 | 1 | 1 | 1 |
|---|---|---|---|

 47H

**Description:** Bits 0-3 of the accumulator are swapped with bits 4-7 of the accumulator.

**Operation:** (A4-7)  $\longleftrightarrow$  (A0-3)

**Example:** Pack bits 0-3 of locations 50-51 into location 50.

```
PCKDIG: MOV R0, #50      ;MOVE '50' DEC TO REG 0
          MOV R1, #51      ;MOVE '51' DEC TO REG 1
          XCHD A,@R0       ;EXCHANGE BITS 0-3 OF ACC
                           ;AND LOCATION 50
          SWAP A           ;SWAP BITS 0-3 AND 4-7 of ACC
          XCHD A,@R1       ;EXCHANGE BITS 0-3 OF ACC AND
                           ;LOCATION 51
          MOV @R0,A        ;MOVE CONTENTS OF ACC TO
                           ;LOCATION 50
```

### XCH A,R<sub>r</sub> Exchange Accumulator-Register Contents

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 0 | 0 | 1 | 0 |
|---|---|---|---|

|   |   |   |   |
|---|---|---|---|
| 1 | r | r | r |
|---|---|---|---|

 28H-2FH

**Description:** The contents of the accumulator and the contents of working register 'r' are exchanged.

**Operation:** (A)  $\longleftrightarrow$  (R<sub>r</sub>) r = 0-7

**Example:** Move PSW contents to Reg 7 without losing accumulator contents.

```
XCHAR7: XCH A,R7        ;EXCHANGE CONTENTS OF REG 7
                           ;AND ACC
          MOV A, PSW      ;MOVE PSW CONTENTS TO ACC
          XCH A,R7        ;EXCHANGE CONTENTS OF REG 7
                           ;AND ACC AGAIN
```



### XCH A,@Rr Exchange Accumulator and Data Memory Contents

**Description:** The contents of the accumulator and the contents of the resident data memory location addressed by bits 0-5\* of register 'r' are exchanged. Register 'r' contents are unaffected.

**Example:** Decrement contents of location 52.

|                   |                                 |
|-------------------|---------------------------------|
| DEC52: MOV R0,#52 | ;MOVE '52' DEC TO ADDRESS REG 0 |
| XCH A,@R0         | ;EXCHANGE CONTENTS OF ACC       |
|                   | ;AND LOCATION 52                |
| DEC A             | ;DECREMENT ACC CONTENTS         |
| XCH A,@R0         | ;EXCHANGE CONTENTS OF ACC       |
|                   | ;AND LOCATION 52 AGAIN          |

**Description:** This instruction exchanges bits 0-3 of the accumulator with bits 0-3 of the data memory location addressed by bits 0-5\* of register 'r'. Bits 4-7 of the accumulator, bits 4-7 of the data memory location, and the contents of register 'r' are unaffected.

**Example:** Assume program counter contents have been stacked in locations 22-23.

```
XCHNIB: MOV R0,#23      ;MOVE '23' DEC TO REG 0
        CLR A            ;CLEAR ACC TO ZEROS
        XCHD A,@R0       ;EXCHANGE BITS 0-3 OF ACC AND
                          ;LOCATION 23 (BITS 8-11 OF PC ARE
                          ;ZEROED. ADDRESS REFERS TO PAGE 0)
```

**Description:** Data in the accumulator is EXCLUSIVE ORed with the mask contained in working register 'r'.

**Example:** XORREG: XRL A,R5 ;'XOR' ACC CONTENTS WITH  
:MASK IN REG 5



## MCS-48 INSTRUCTION SET

---

### XRL A,@Rr Logical XOR Accumulator With Memory Mask

---

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 1 | 0 | 1 |
| 0 | 0 | 0 | r |

 D0H-D1H

**Description:** Data in the accumulator is EXCLUSIVE ORed with the mask contained in the data memory location addressed by register 'r', bits 0-5\*.

**Operation:**  $(A) \leftarrow (A) \text{ XOR } ((Rr))$   $r = 0-1$

**Example:** XORDM: MOV R1, #20H ;MOVE '20' HEX TO REG 1  
XRL A,@R1 ;'XOR' ACC CONTENTS WITH MASK  
;IN LOCATION 32

### XRL A,#data Logical XOR Accumulator With Immediate Mask

---

**Encoding:**

|   |   |   |   |
|---|---|---|---|
| 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 |

 d7 d6 d5 d4 d3 d2 d1 d0 D3H

**Description:** This is a 2-cycle instruction. Data in the accumulator is EXCLUSIVE ORed with an immediately-specified mask.

**Operation:**  $(A) \leftarrow (A) \text{ XOR data}$

**Example:** XORID: XOR A,#HEXTEN ;XOR CONTENTS OF ACC WITH MASK  
;EQUAL VALUE OF SYMBOL 'HEXTEN'



---

# **MCS<sup>®</sup>-48 Application Examples**

---

**5**



# CHAPTER 5

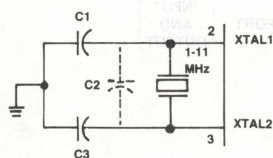
## MCS<sup>®</sup>-48 APPLICATION EXAMPLES

### 5.0 INTRODUCTION

This chapter is organized in two sections, Hardware and Software. The hardware section gives examples of some typical configurations of MCS-48 components, while the software section gives assembly language listings of some common applications routines.

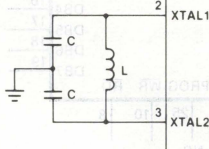
### 5.1 HARDWARE EXAMPLES

#### OSCILLATOR MODE



$C1 = 5\text{pf} \pm \frac{1}{2}\text{pf} + (\text{STRAY} < 5\text{pf})$   
 $C2 = (\text{CRYSTAL} + \text{STRAY}) < 8\text{pf}$   
 $C3 = 20\text{pf} \pm 1\text{pF} + (\text{STRAY} < 5\text{pf})$

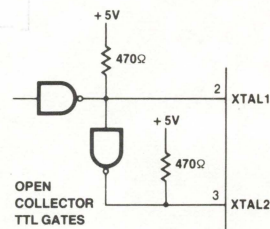
#### LC OSCILLATOR MODE NOT APPLICABLE TO 8021H/8022



| L                 | C     | NOMINAL f |
|-------------------|-------|-----------|
| 45 $\mu\text{H}$  | 20 pF | 5.2 MHz   |
| 120 $\mu\text{H}$ | 20 pF | 3.2 MHz   |

$f = \frac{1}{2\pi\sqrt{LC}}$   
 $C' = \frac{C + 3C_{pp}}{2}$   
 $C_{pp} \sim 5\text{-}10\text{ pF}$   
 PIN-TO-PIN CAPACITANCE

#### DRIVING FROM EXTERNAL SOURCE



CRYSTAL PARALLEL RESISTANCE SHOULD BE LESS THAN 75  $\Omega$  AT 11 MHz, LESS THAN 180  $\Omega$  AT 3.6 MHz. EACH C SHOULD BE APPROXIMATELY 20pF INCLUDING STRAY CAPACITANCE

FOR THE 8049H XTAL1 MUST BE HIGH 35-65% OF THE PERIOD AND XTAL2 MUST BE HIGH 35-65% OF THE PERIOD.

RISE AND FALL TIMES MUST NOT EXCEED 20ns.

Intel Corporation Assumes No Responsibility for the Use of any Circuitry Other Than Circuitry Embodied in an Intel Product. No Other Circuit Patent Licenses are Implied.

Figure 5-1. Suggested Circuits



## MCS-48 APPLICATION EXAMPLES

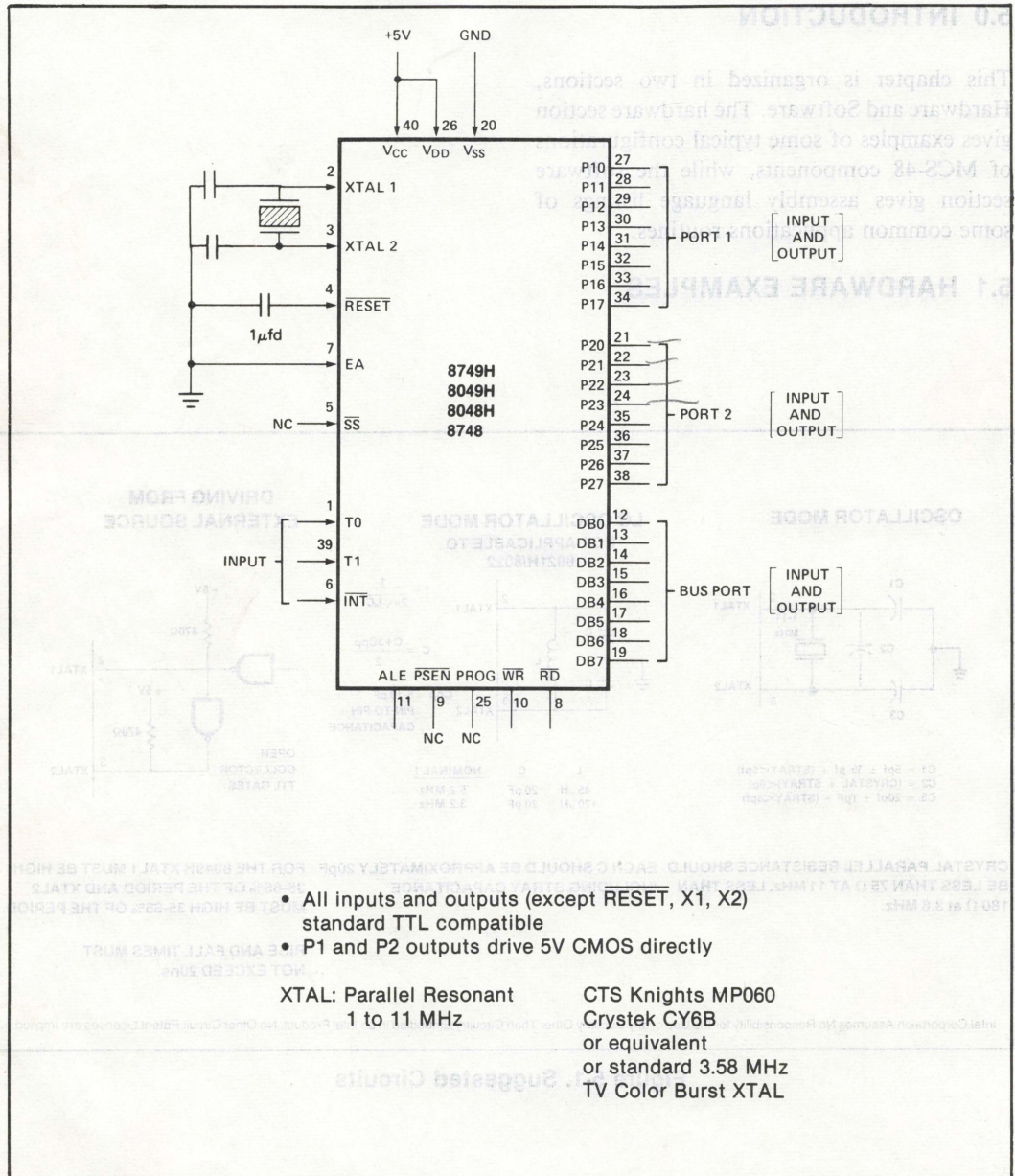


Figure 5-2. The Stand Alone 8048H/8049H/8748H/8749H



## MCS-48 APPLICATION EXAMPLES

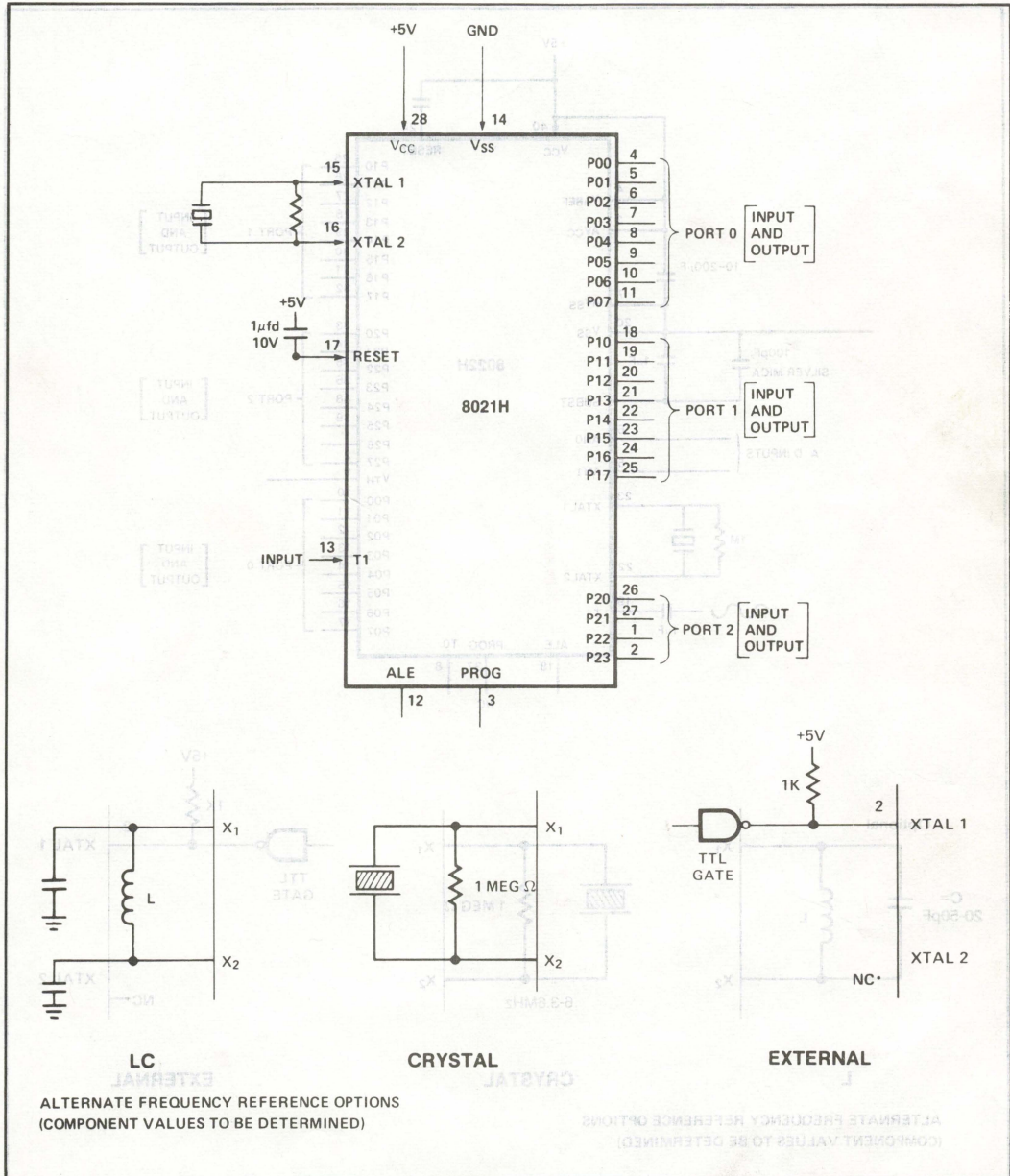
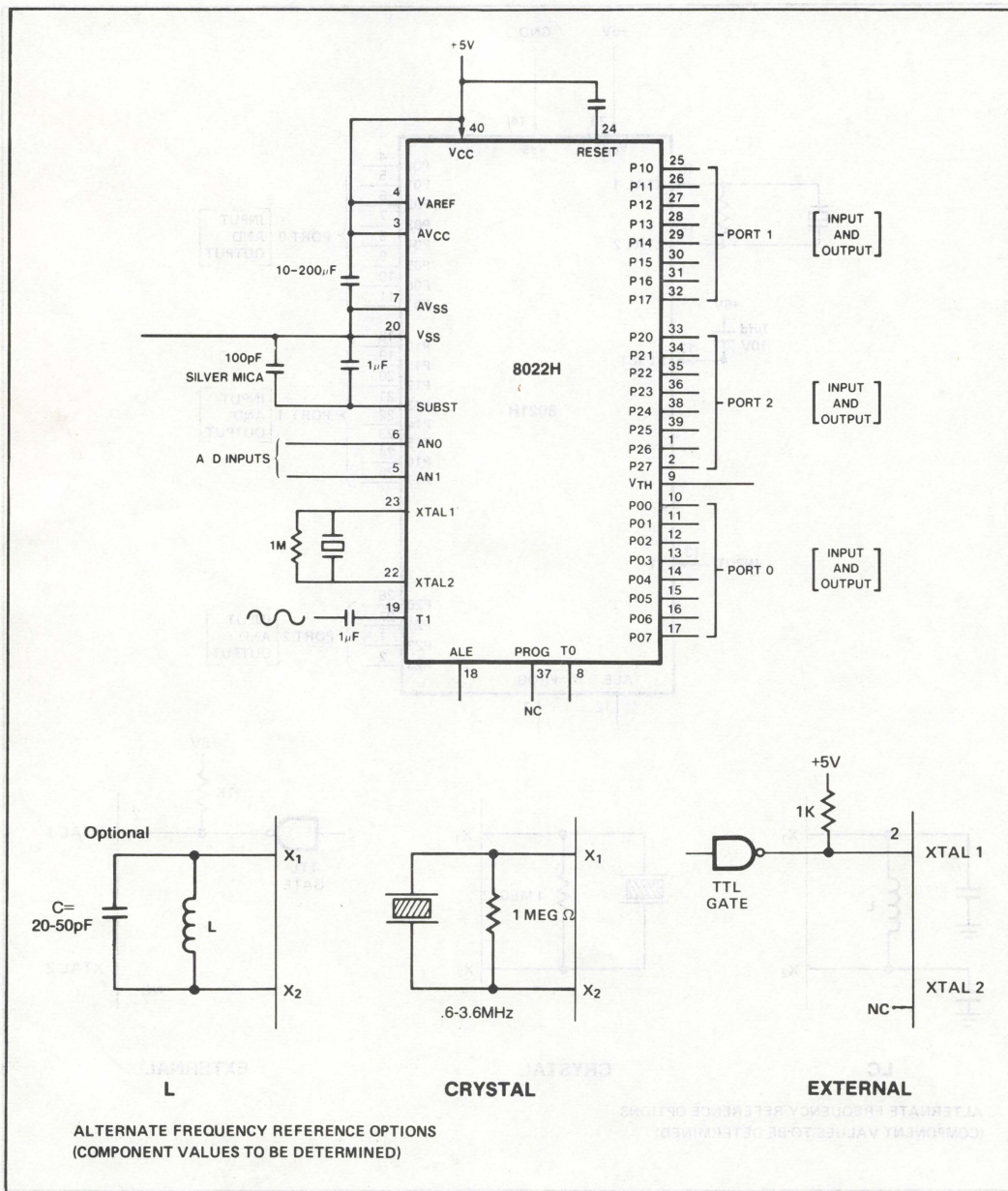


Figure 5-3. The Stand Alone 8021H



## MCS-48 APPLICATION EXAMPLES



**Figure 5-4. The Stand Alone 8022H**



## MCS-48 APPLICATION EXAMPLES

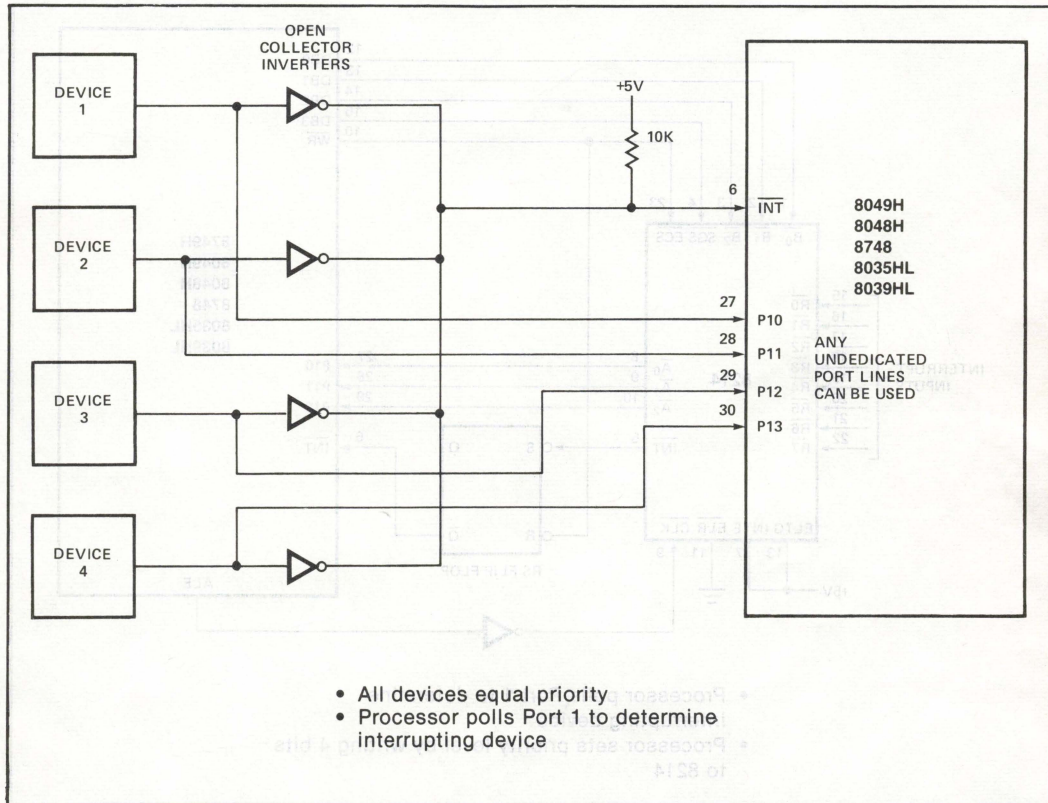
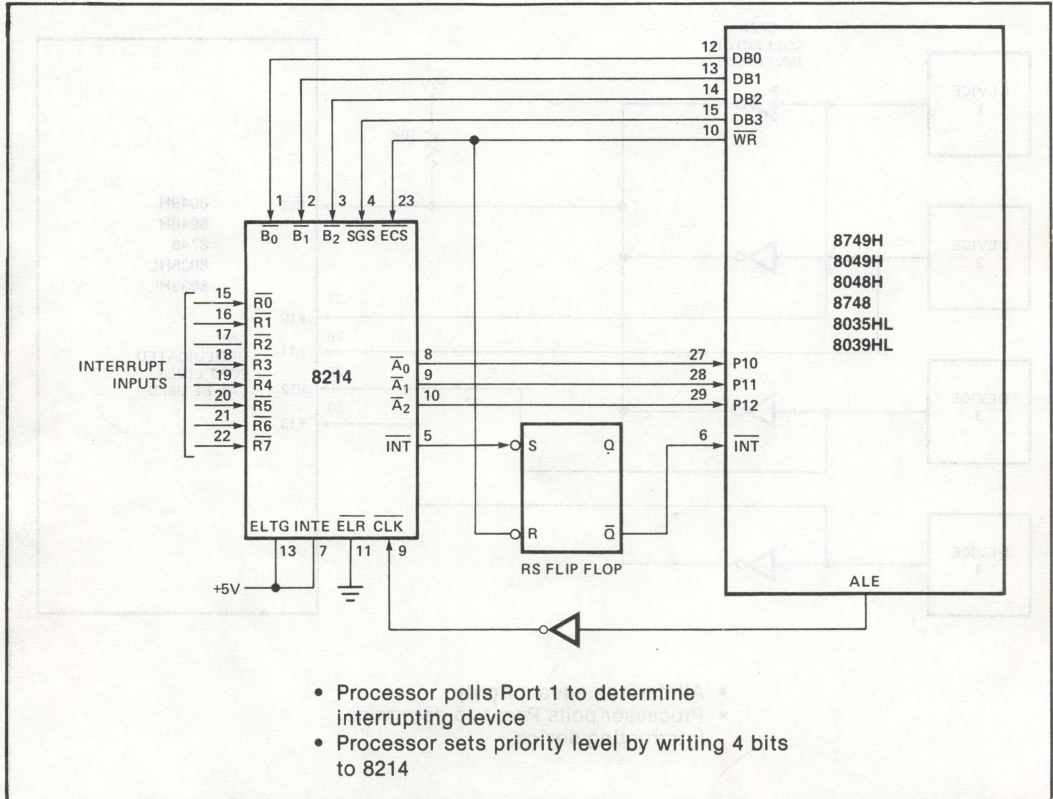


Figure 5-5. Multiple Interrupt Sources



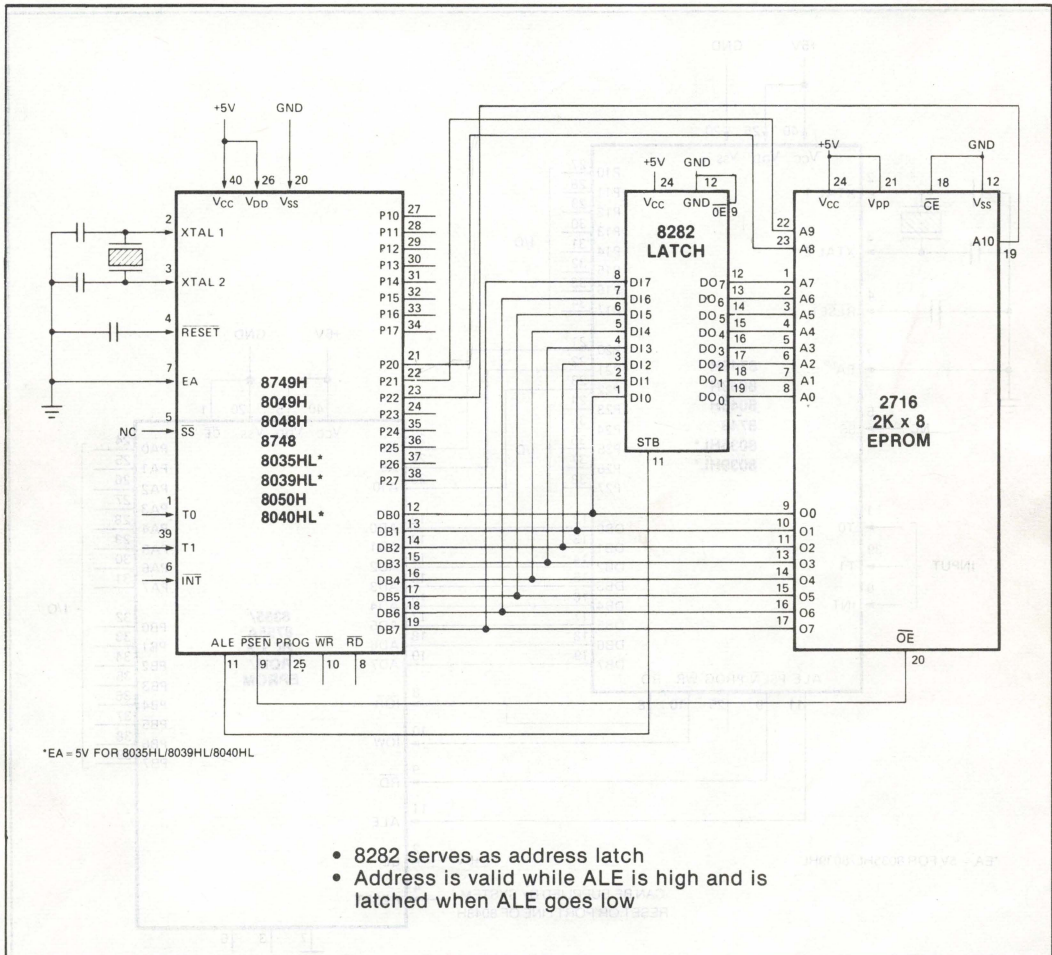
## MCS-48 APPLICATION EXAMPLES



**Figure 5-6. Multiple Interrupts with Priority Levels**



## MCS-48 APPLICATION EXAMPLES



**Figure 5-7. External Program Memory**



## MCS-48 APPLICATION EXAMPLES

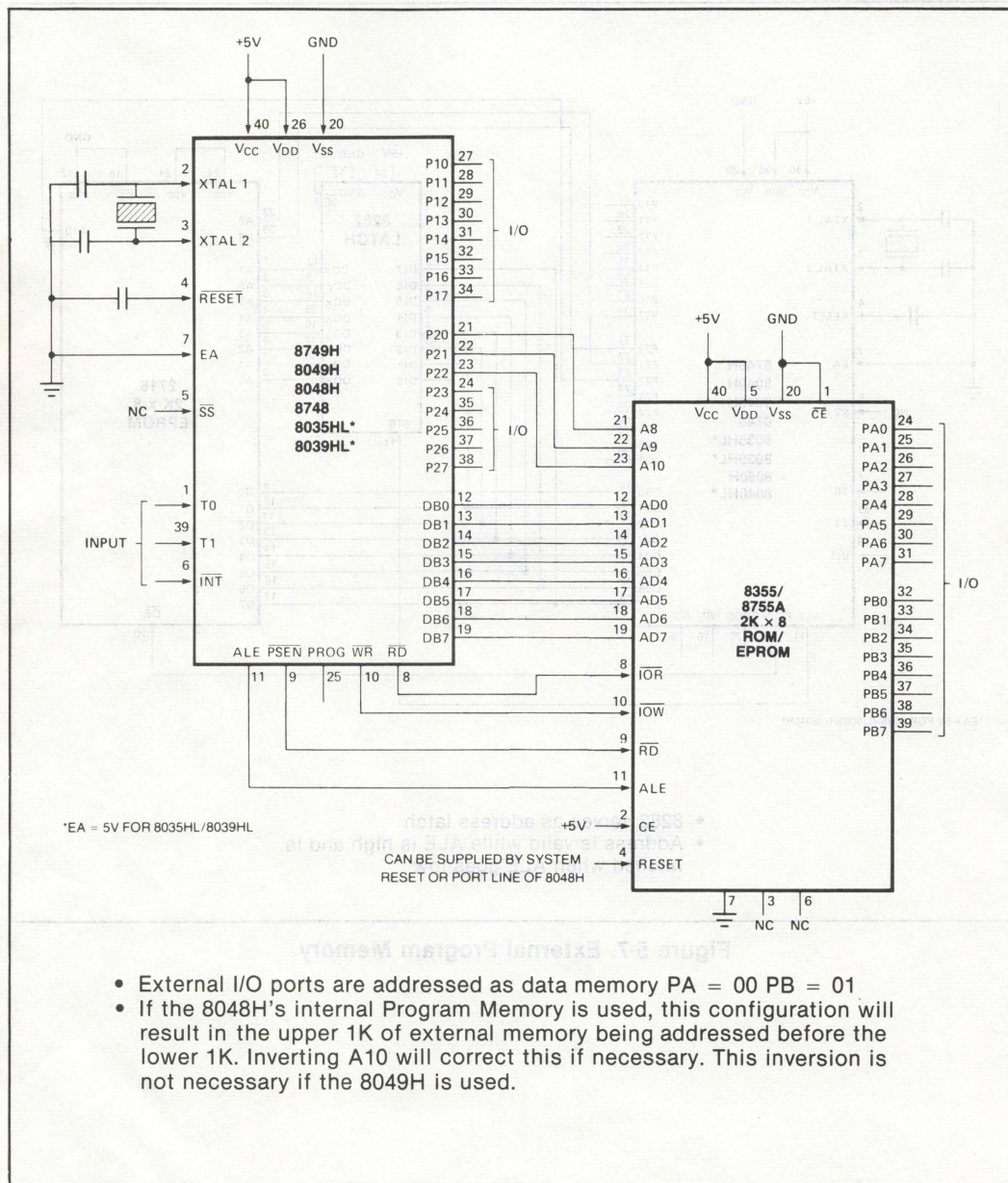
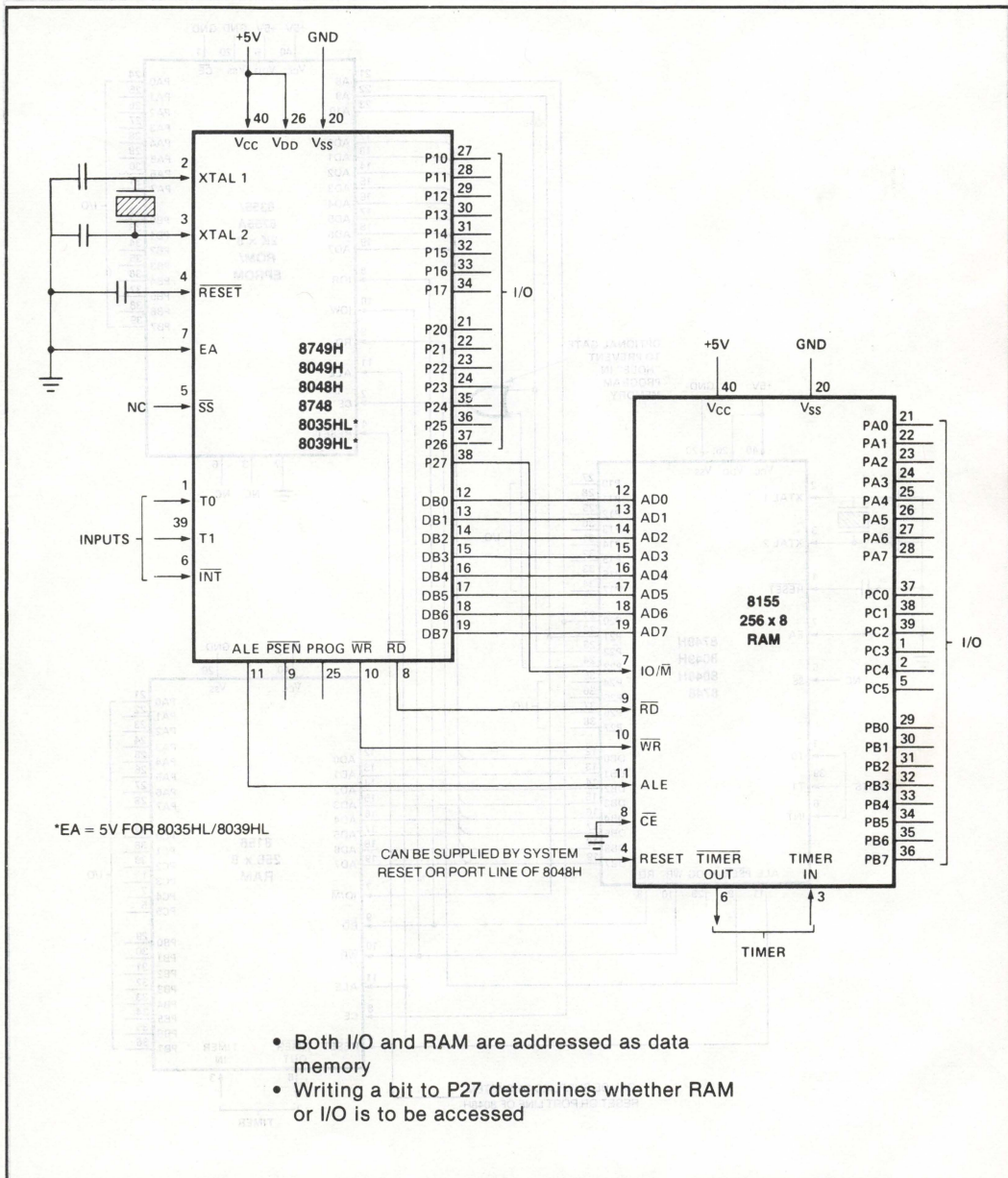


Figure 5-8. Adding a Program Memory and I/O Expander



## MCS-48 APPLICATION EXAMPLES



- Both I/O and RAM are addressed as data memory
- Writing a bit to P27 determines whether RAM or I/O is to be accessed

Figure 5-9. Adding a Data Memory and I/O Expander



## MCS-8 APPLICATION EXAMPLES

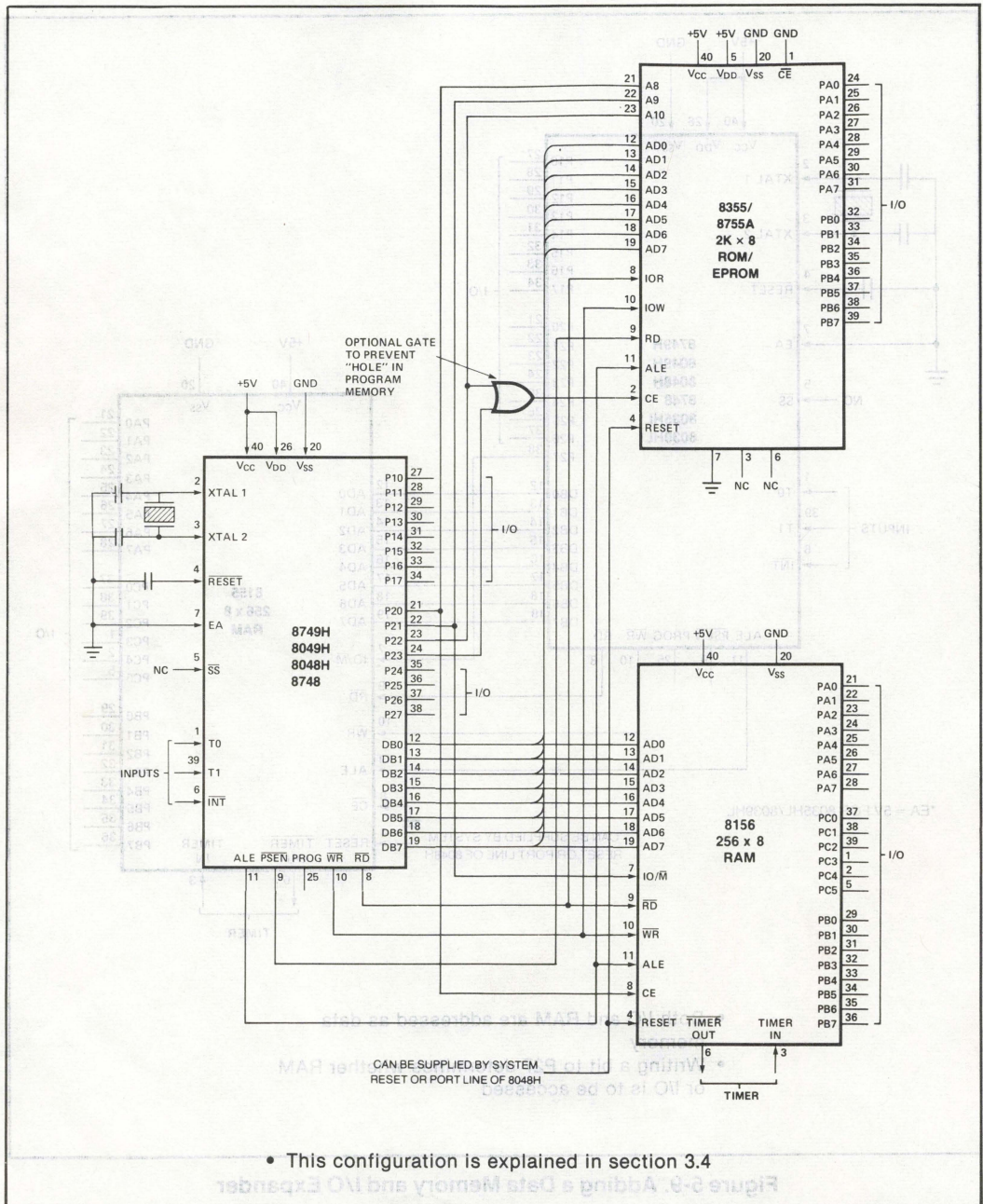


Figure 5-10. The Three-Chip System







## MCS-48 APPLICATION EXAMPLES

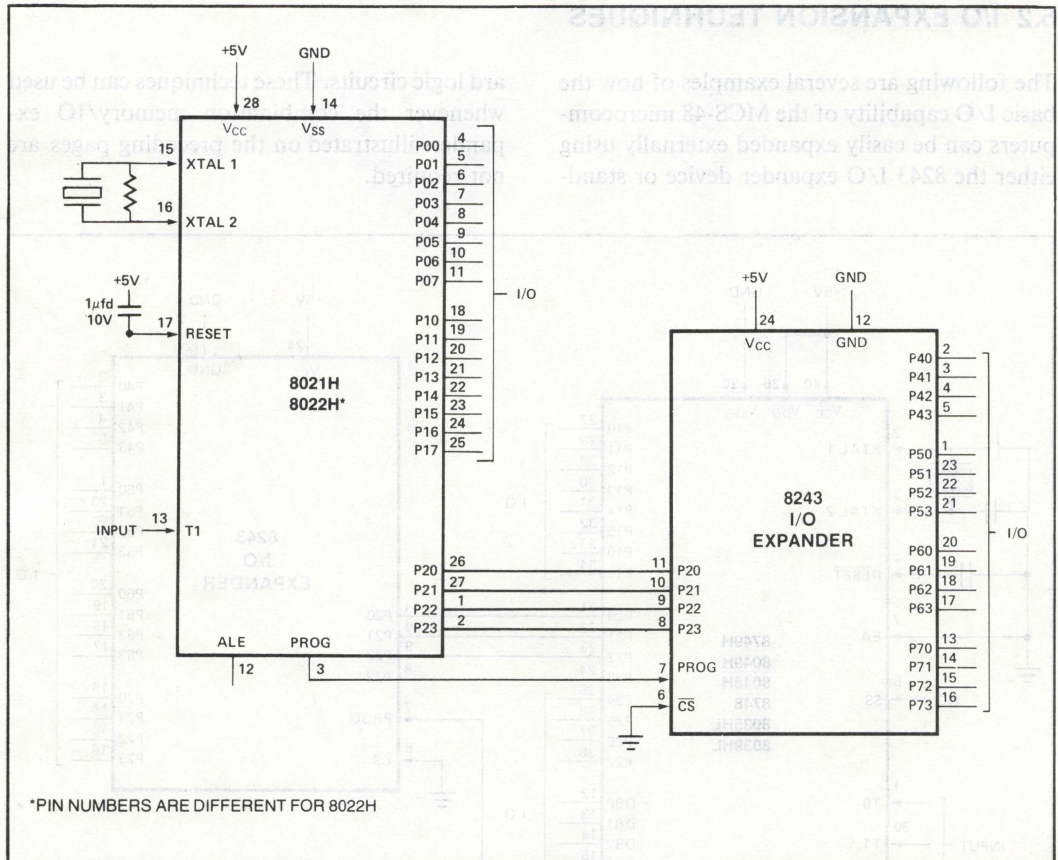


Figure 5-12. Adding an I/O Expander to the 8021H, 8022H



## MCS-48 APPLICATION EXAMPLES

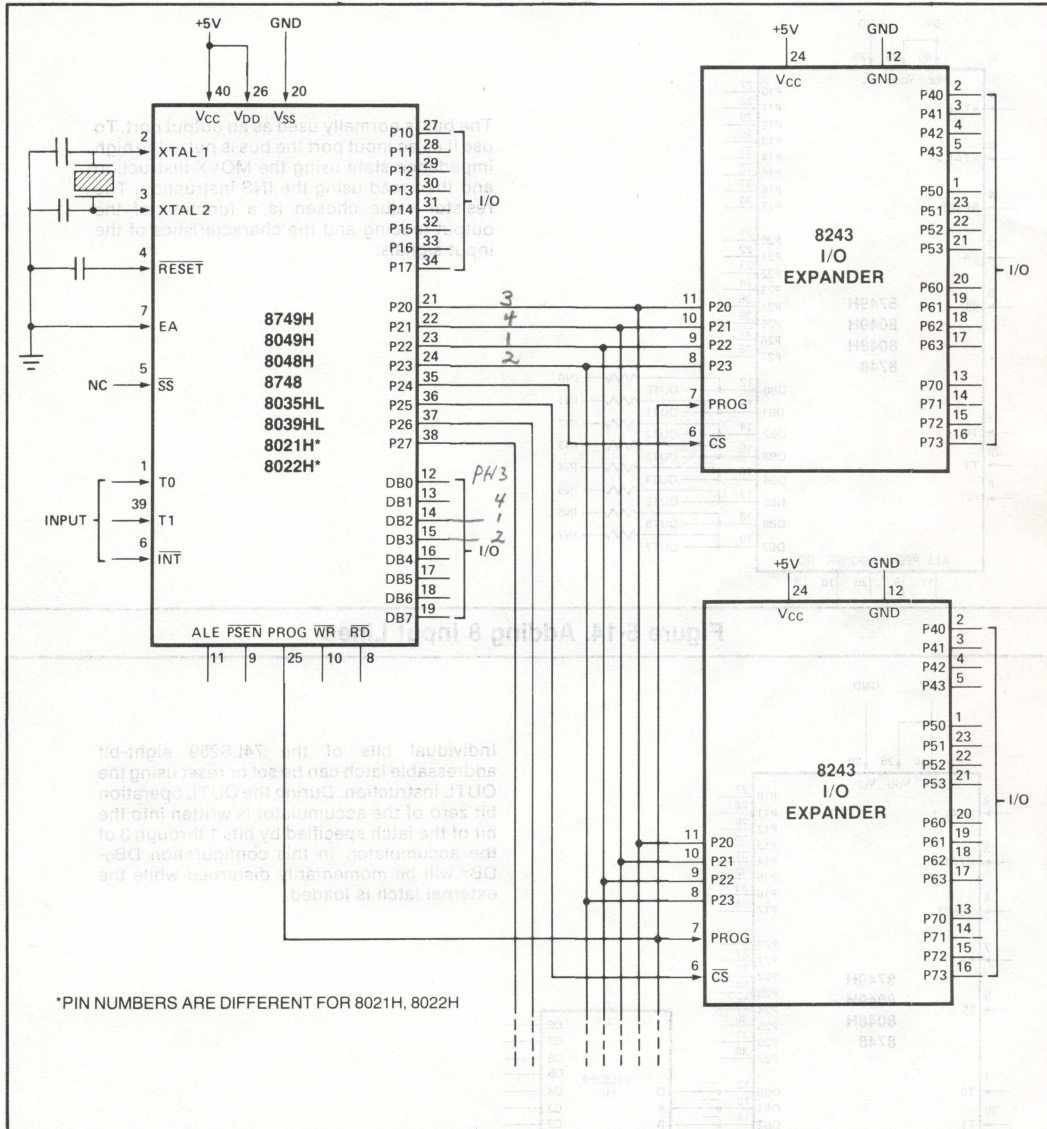


Figure 5-13. Adding Multiple I/O Expanders



## MCS-48 APPLICATION EXAMPLES

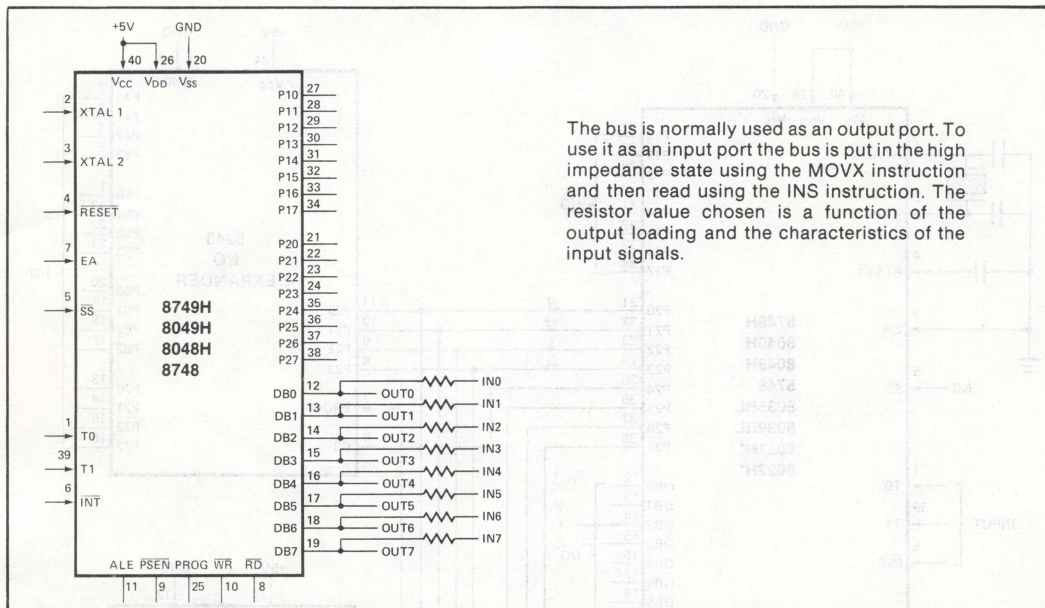


Figure 5-14. Adding 8 Input Lines

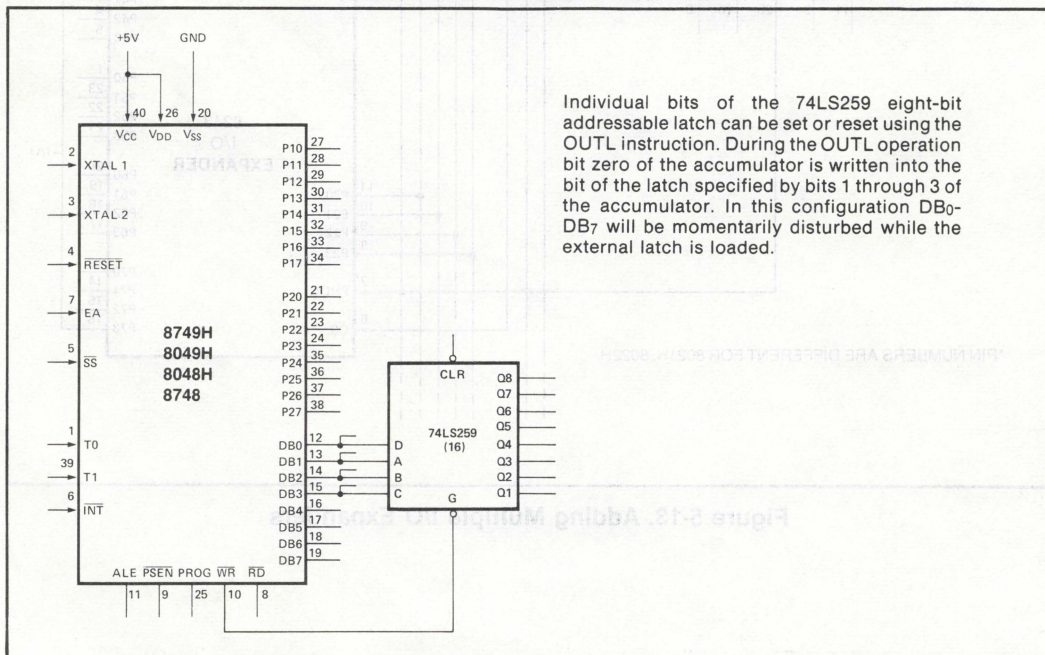


Figure 5-15. Adding 8 Output Lines



## MCS-48 APPLICATION EXAMPLES

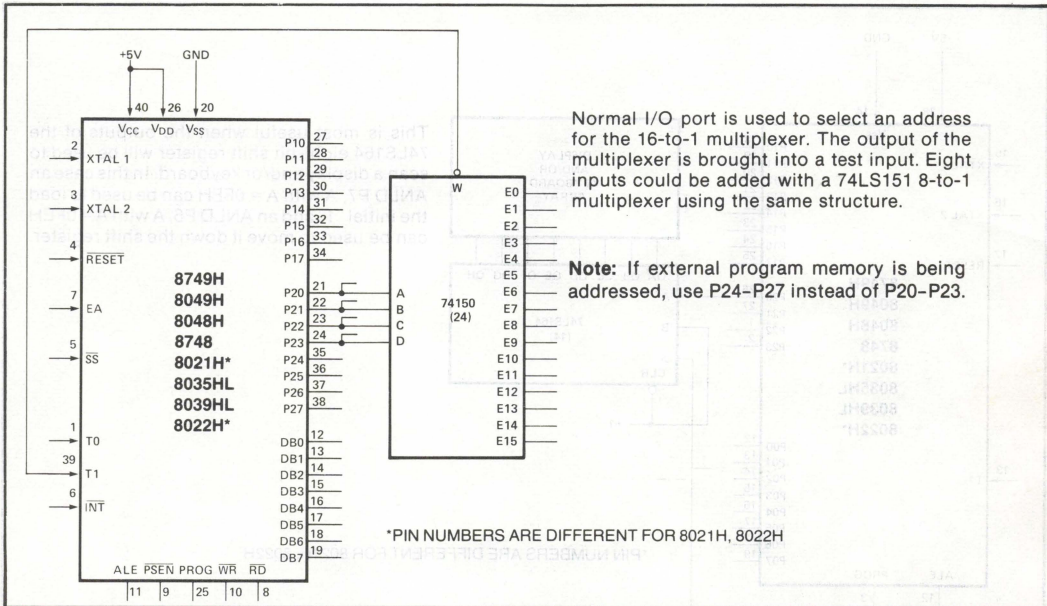


Figure 5-16. Adding 16 Input Lines

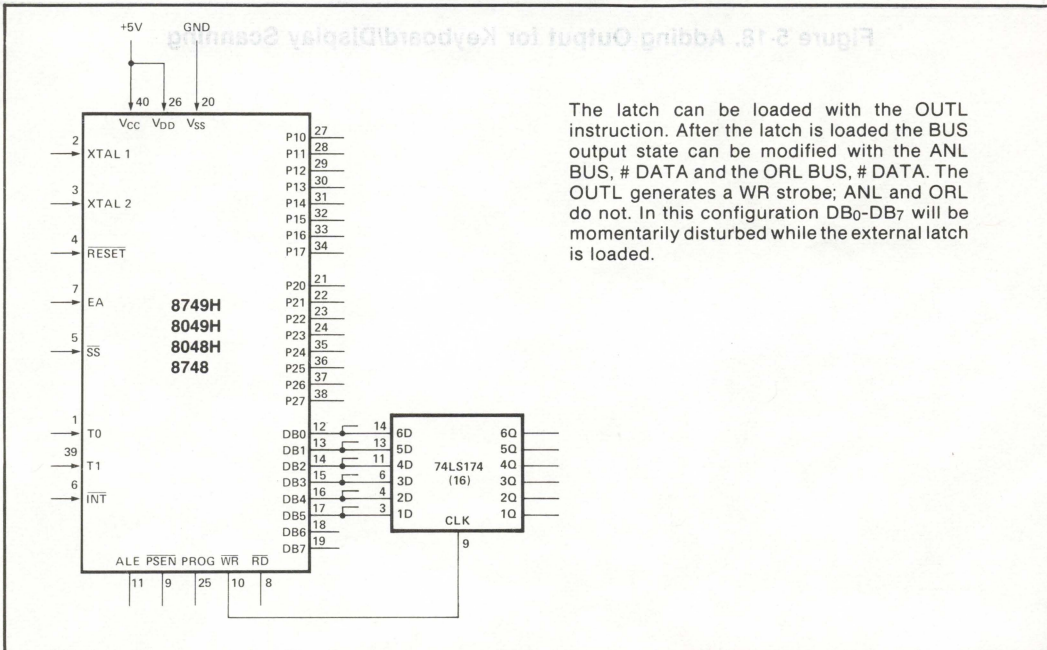


Figure 5-17. Adding 6 Output Lines



## MCS-48 APPLICATION EXAMPLES

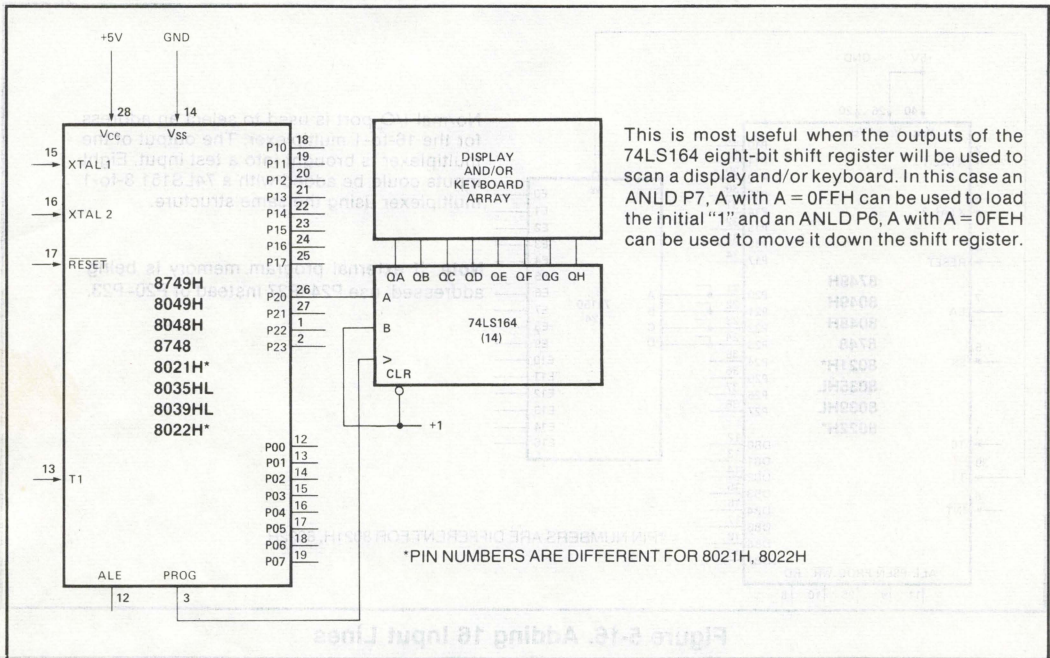
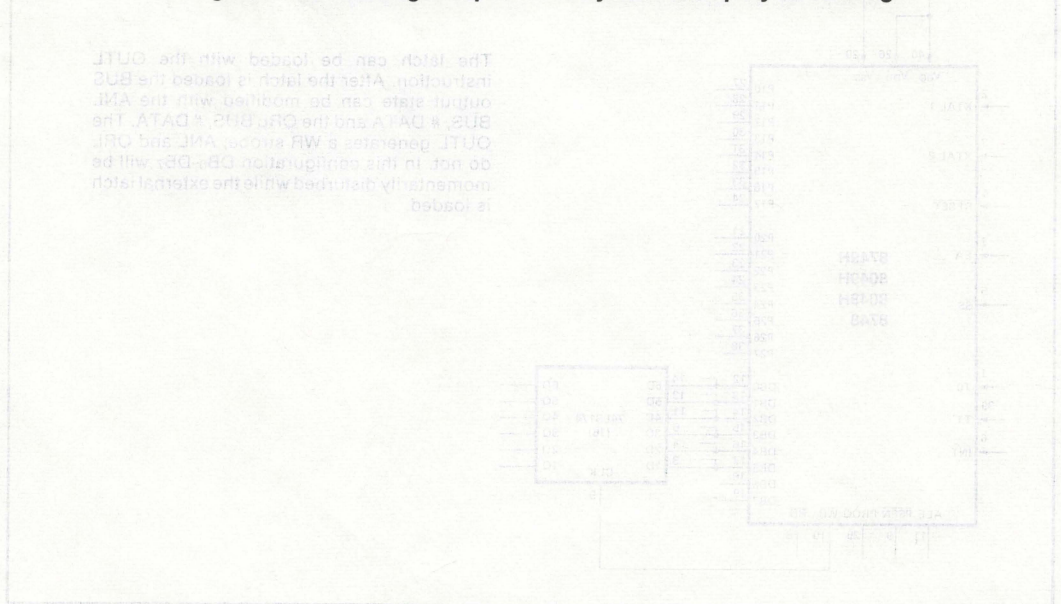


Figure 5-18. Adding Output for Keyboard/Display Scanning





## MCS-48 APPLICATION EXAMPLES

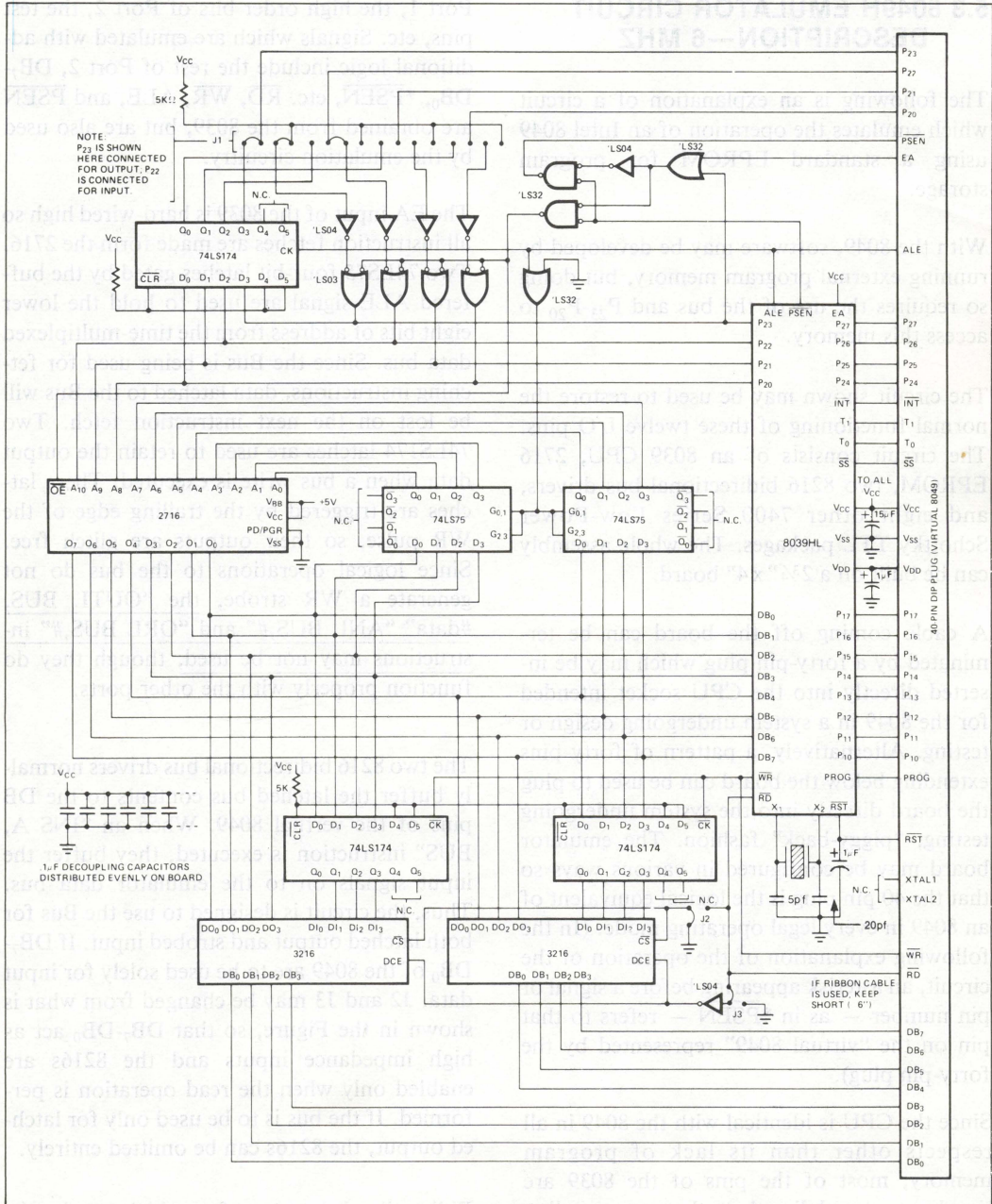


Figure 5-19. 8049H Emulator Circuit



### 5.3 8049H EMULATOR CIRCUIT DESCRIPTION—6 MHZ

The following is an explanation of a circuit which emulates the operation of an Intel 8049 using a standard EPROM for program storage.

With the 8049, software may be developed by running external program memory, but doing so requires the use of the bus and  $P_{23}$ - $P_{20}$  to access this memory.

The circuit shown may be used to restore the normal functioning of these twelve I/O pins. The circuit consists of an 8039 CPU, 2716 EPROM, two 8216 bidirectional bus drivers, and eight other 7400 Series Low-Power Schottky TTL packages. The whole assembly can be built on a  $2\frac{3}{4} \times 4$ " board.

A cable coming off the board can be terminated by a forty-pin plug which may be inserted directly into the CPU socket intended for the 8049 in a system undergoing design or testing. Alternatively, a pattern of forty pins extending below the board can be used to plug the board directly into the system undergoing testing, "piggy-back" fashion. The emulator board may be configured in various ways so that the 40 pin plug is the logical equivalent of an 8049 in every legal operating mode. (In the following explanation of the operation of the circuit, an asterisk appearing before a signal or pin number — as in  $*PSEN$  — refers to that pin on the "virtual 8049" represented by the forty-pin plug).

Since the CPU is identical with the 8049 in all respects other than its lack of program memory, most of the pins of the 8039 are simply connected directly to the corresponding pins of the forty-pin plug. These include all of

Port 1, the high order bits of Port 2, the test pins, etc. Signals which are emulated with additional logic include the rest of Port 2,  $DB_7$ - $DB_0$ ,  $*PSEN$ , etc.  $\overline{RD}$ ,  $\overline{WR}$ ,  $ALE$ , and  $\overline{PSEN}$  are obtained from the 8039, but are also used by the emulation circuitry.

The EA input of the 8039 is hard-wired high so all instruction fetches are made from the 2716. Two 74LS75 four-bit latches gated by the buffered  $ALE$  signal are used to hold the lower eight bits of address from the time-multiplexed data bus. Since the Bus is being used for fetching instructions, data latched to the Bus will be lost on the next instruction fetch. Two 74LS174 latches are used to retain the output data when a bus write is executed. These latches are triggered by the trailing edge of the  $\overline{WR}$  pulse, so their outputs are glitch free. Since logical operations to the bus do not generate a  $\overline{WR}$  strobe, the "OUTL BUS, #data" "ANL BUS, #" and "ORL BUS, #" instructions may not be used, though they do function properly with the other ports.

The two 8216 bidirectional bus drivers normally buffer the latched bus contents to the  $DB$  pins of the virtual 8049. When an "INS A, BUS" instruction is executed, they buffer the input signals on to the emulator data bus. Thus, the circuit is designed to use the Bus for both latched output and strobed input. If  $DB_7$ - $DB_0$  of the 8049 are to be used solely for input data, J2 and J3 may be changed from what is shown in the Figure, so that  $DB_7$ - $DB_0$  act as high impedance inputs and the 8216s are enabled only when the read operation is performed. If the bus is to be used only for latched output, the 8216s can be omitted entirely.

Bidirectional data transfers which require the transfer of address information as well as



## MCS-48 APPLICATION EXAMPLES

data, such as to and from external data memory, require removal of the 8216s and replacement with 16-pin jumper blocks on which the  $DB_x$  pins are connected with the respective  $DO_x$  pins.

The lower four bits of Port 2 are also used in fetching instructions from the 2716, in addition to their use in input or output pins in the user's system. In configuring the emulator for a particular application, the user must dedicate each of these as either an input or output pin and connect jumper set J1 accordingly. Any mix of input and output pins is allowed. At the beginning of each instruction fetch, the last data written to P2 will be present on  $P_{23}$ - $P_{20}$  at the rising edge of ALE and will be latched by a 74LS174. The latched data may be connected through the jumpers to those pins which will be used as outputs on the 8049. Emulator pins used as inputs should be pulled above 2.0V for a logic "one". If this is not the case, i.e., if switches to Ground are to be read, 50K pullup resistors should be added to the circuit on each input. They were omitted from the diagram to minimize input loading.

Pins which will be used as inputs may be connected to the input of an OR gate formed of inverters and open-collector NAND gates. The input signals will be relayed directly to the 8039 and will be read by an "IN A,P2" instruction. But when PSEN is low, the NAND outputs are forced off, allowing the 8039 pins to be used for high-order program addressing. Open-collector outputs are needed to prevent line contention when PSEN is not low.

If 8243s will be used in the final system, the low order pins of Port 2 must be connected directly to the plug. This may be done by replacing the Port 2 latch with four jumpers connecting the inputs to the outputs. The

NANDs should be removed or disabled by grounding the common NAND inputs.

The cluster of three OR gates is used to enable the on-board 2716 and generate the \*PSEN signal, each of which is a function of PSEN, \*EA, and the high order bit of the program counter. Thus \*PSEN is generated, forcing an off-board read, only when a jump has been made to Memory Bank 1 or when \*EA is brought high. If this feature is to be used to address off-board memory,  $DB_7$ - $DB_0$  may not be used for normal I/O. The 8216s and 74LS174 must be replaced with jumper blocks and the open collector NAND gates disabled, as explained above. The same changes are required to operate the board in single step mode.



## MCS-48 APPLICATION EXAMPLES

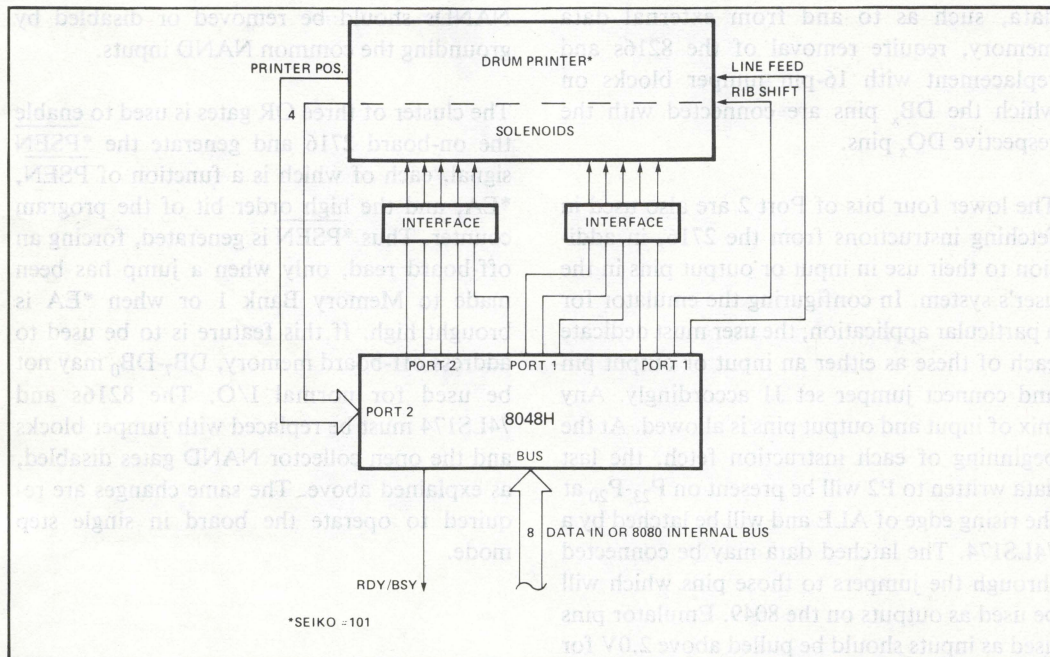


Figure 5-20. 8048H Interface to Drum Printer

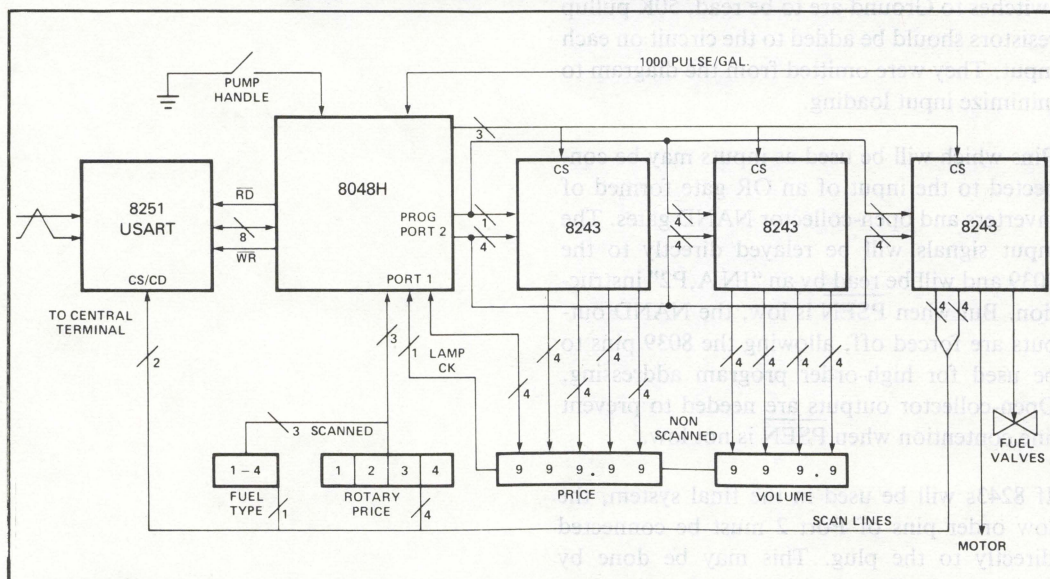


Figure 5-21. MCS-48 Gas Pump



## MCS-48 APPLICATION EXAMPLES

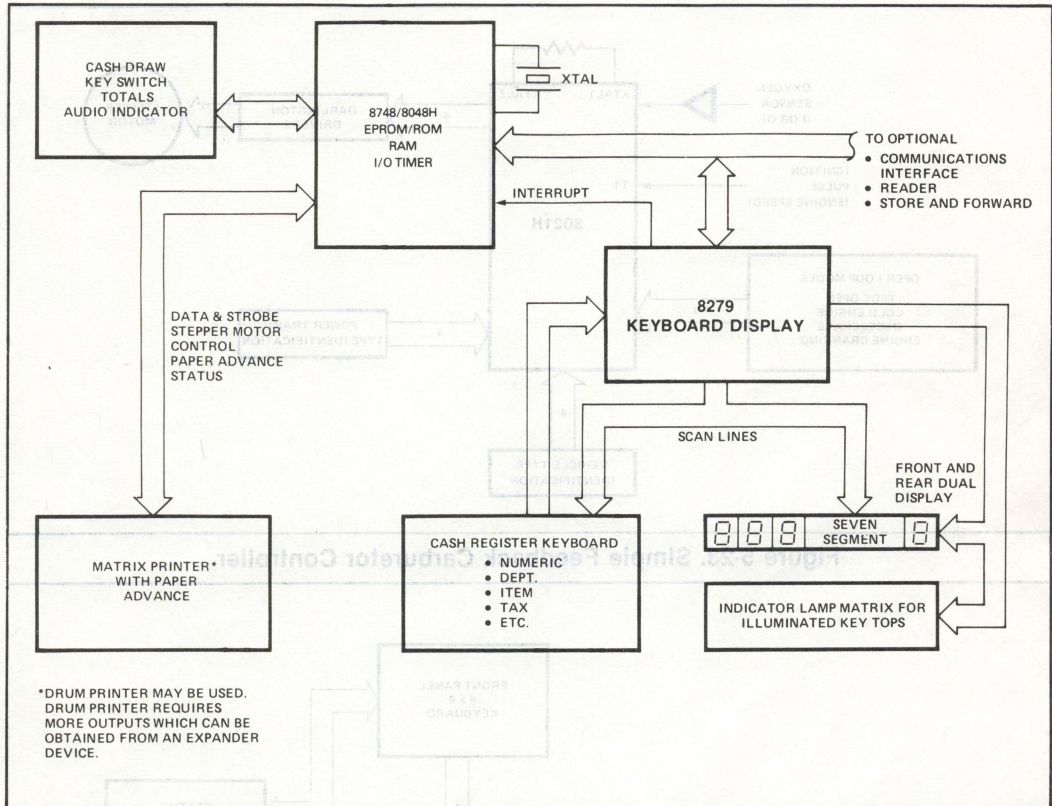


Figure 5-22. Low Cost Point of Sale Terminal

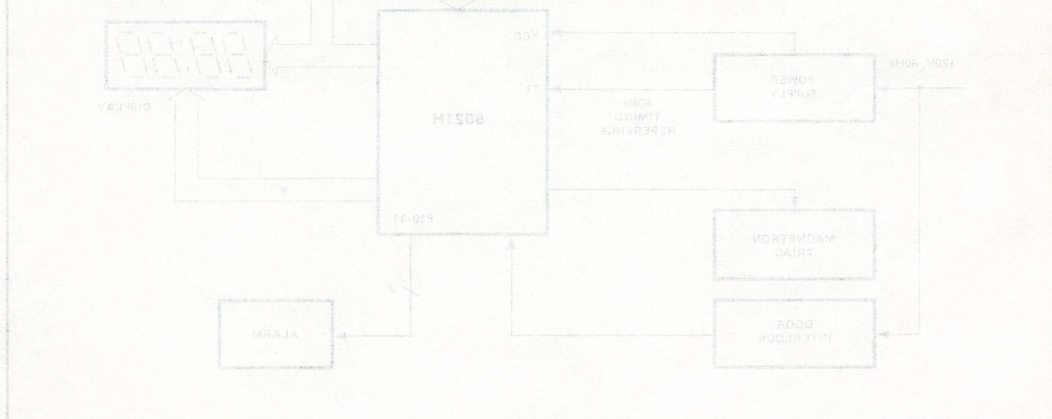


Figure 5-24. Microwave Oven Controller



## MCS-48 APPLICATION EXAMPLES

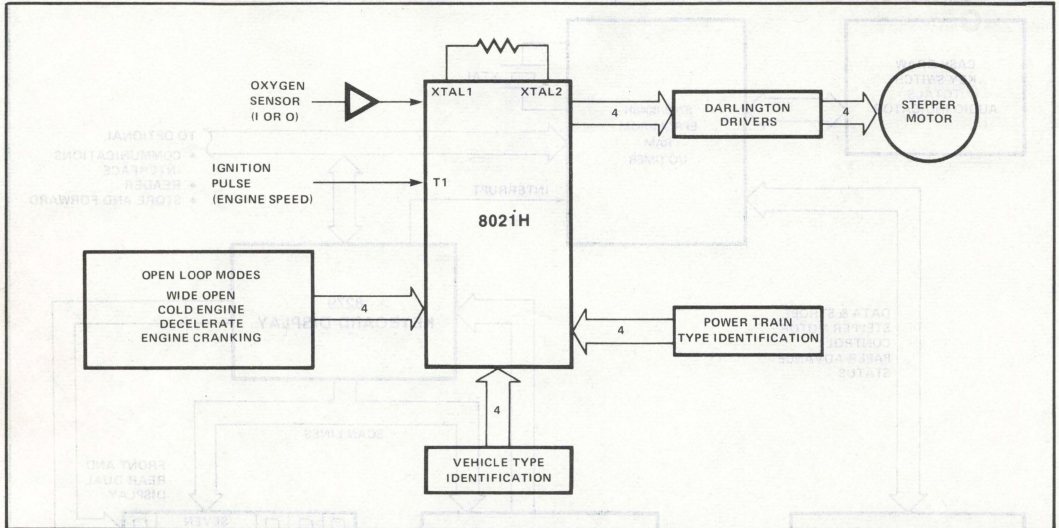


Figure 5-23. Simple Feedback Carburetor Controller

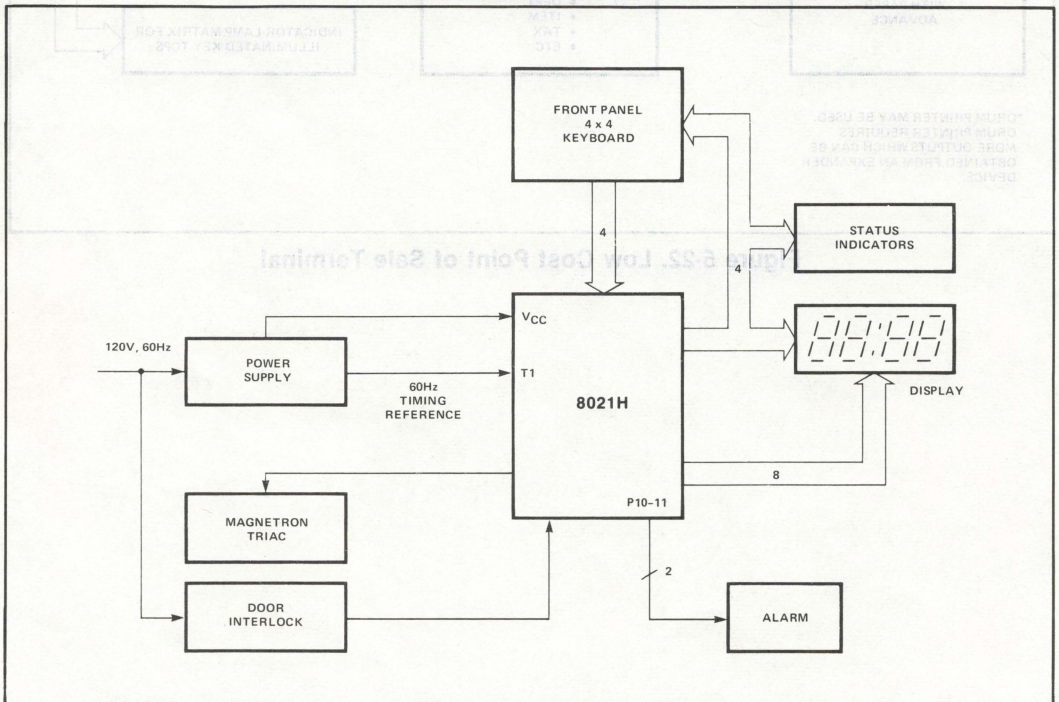


Figure 5-24. Microwave Oven Controller



## MCS-48 APPLICATION EXAMPLES

### 5.4 SOFTWARE EXAMPLES

The following routines are written as subroutines. R0 and R1 are used as data

pointers, R2 is used as an extension of the accumulator and R3 is used as a loop counter.

RX0 = R0

AEX = R2

#### DOUBLE ADD

```
DADD:  DEC  RX0      ;GET LOW BYTE AND ADD TO A
        ADD  A,@RX0
        INC  RX0      ;GET HI BYTE AND ADD TO AEX
        XCH  A,AEX
        ADDC A,@RX0
        XCH  A,AEX
        RET          ;RETURN
```

#### DOUBLE SUBTRACT

```
DMIN:  DEC  RX0      ;GET LOW BYTE AND SUB FROM A
        CPL  A
        ADD  A,@RX0
        CPL  A
        INC  RX0      ;GET HI BYTE AND SUB FROM AEX
        XCH  A,AEX
        CPL  A
        ADDC A,@RX0
        CPL  A
        XCH  A,AEX
        RET          ;RETURN
```

#### DOUBLE LOAD

```
DLD:   DEC  RX0      ;GET LOW BYTE AND PLACE IN A
        MOV  A,@RX0
        INC  RX0      ;GET HI BYTE AND PLACE IN AEX
        XCH  A,AEX
        MOV  A,@RX0
        XCH  A,AEX
        RET          ;RETURN
```

#### DOUBLE STORE

```
DST:   DEC  RX0      ;MOVE A INTO LOW BYTE
        MOV  @RX0,A
        INC  RX0      ;MOVE AEX INTO HIGH BYTE
        XCH  A,AEX
        MOV  @RX0,A
        XCH  A,AEX
        RET          ;RETURN
```



## MCS-48 APPLICATION EXAMPLES

### DOUBLE EXCHANGE

```

DEX:    DEC    RX0      ;EXCHANGE A AND LOW BYTE
        XCH    A,@RX0
        INC    RX0      ;EXCHANGE AEX AND HIGH BYTE
        XCH    A,AEX
        XCH    A,@RX0
        XCH    A,AEX
        RET          ;RETURN
    
```

### DOUBLE LEFT LOGICAL SHIFT

```

LLSH:   RLC    A        ;SHIFT A
        XCH    A,AEX    ;SHIFT AEX
        RLC    A
        XCH    A,AEX
        RET          ;RETURN
    
```

### DOUBLE RIGHT LOGICAL SHIFT

```

RLSH:   XCH    A,AEX    ;SHIFT AEX
        RRC    A
        XCH    A,AEX
        RRC    A        ;SHIFT A
        RET          ;RETURN
    
```

### DOUBLE RIGHT ARITHMETIC SHIFT

```

RASH:   CLR    C        ;SET CARRY
        CPL    C
        XCH    A,AEX    ;IF AEX[7] < > 1 THEN
        JB7    $+3
        CLR    C        ;CLEAR CARRY
        RRC    A        ;SHIFT C INTO AEX
        XCH    A,AEX
        RRC    A        ;SHIFT A
        RET          ;RETURN
    
```

### 5.4.1 Single Precision Binary Multiply

This routine assumes a one-byte multiplier and a one-byte multiplicand. The product, therefore, is two-bytes long.

The algorithm follows these steps:

- 1) The registers are arranged as follows:

```

ACC — 0
R1 — Multiplier
R2 — Multiplicand
R3 — Loop Counter (=8)
    
```

The Accumulator and register R1 are treated as a register pair when they are shifted right (see Step 2)

- 2) The Accumulator and R1 are shifted right one place, thus the LSB of the multiplier goes into the carry.
- 3) The multiplicand is added to the accumulator if the carry bit is a 'one'. No action if the carry is a 'zero'.
- 4) Decrement the loop counter and loop (return to Step 2) until it reaches zero.
- 5) Shift the result right one last time just before exiting the routine.

\*The result will be found in the Accumulator (MS Byte) and R1 (LS Byte).



## MCS-48 APPLICATION EXAMPLES

### BINARY MULTIPLY

```

BMPY:  MOV    R3,#08H    ;SET COUNTER TO 8
        CLR   A          ;CLEAR A
        CLR   C          ;CLEAR CARRY BIT

BMPI:   RRC    A          ;DOUBLE SHIFT RIGHT ACC & R1
        XCH   A,R1       ;INTO CARRY
        RRC    A
        XCH   A,R1
        JNC   BMP3       ;IF CARRY = 1 ADD, OTHERWISE DON'T
        ADD   A,R2       ;ADD MULTIPLICAND TO ACCUMULATOR

BMP3:   DJNZ   R3,BMPI    ;DECREMENT COUNTER AND LOOP IF 0
        RRC    A          ;DO A FINAL RIGHT SHIFT AT THE
        XCH   A,R1       ;END OF THE ROUTINE
        RRC    A
        XCH   A,R1
    
```

### 5.4.2 Interrupt Handling

This interrupt routine assumes single level interrupt. The purpose is to store the status of the machine at the time the interrupt occurs by storing contents of all registers, accumulator,

and the status word. At the end of the interrupt the state of the machine is restored and interrupts are enabled again.

```

INTRPT: SELA   RB1       ;SAVE WORKING REGISTERS
        MOV    @R0,A     ;R0 IN ALTERNATE REGISTER
                        ;BANK CONTAINS SACC
                        ;POINTER FOR SAVING
                        ;ACCUMULATOR
        |         |
        |         |     } INTERRUPT SERVICE
        |         |     } ROUTINE
        MOV    R0,SACC    ;RESTORE SACC
        MOV    A,@R0     ;RESTORE ACCUMULATOR
        RETR           ;RESTORE WORKING REGISTERS
                        ;RESTORE PSW AND
                        ;RE-ENABLE INTERRUPTS
    
```



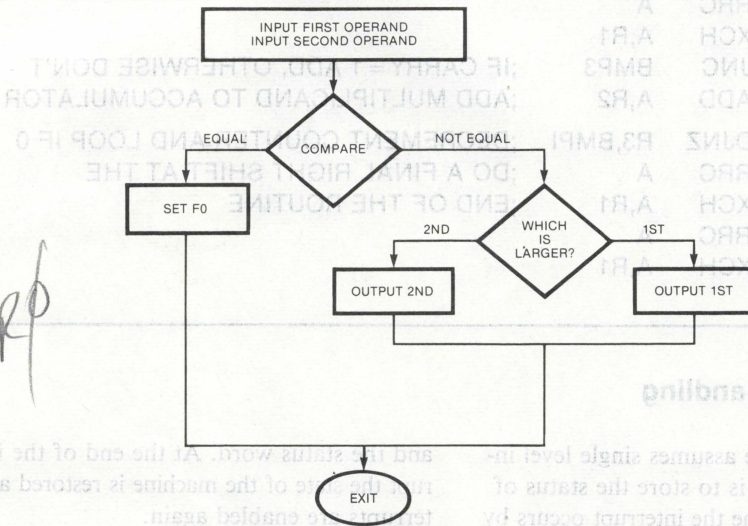
## MCS-48 APPLICATION EXAMPLES

### 5.4.3 Two-Byte Processing System

A suggested model of a processing routine takes two single byte inputs from different ports, compares them, and performs the following, depending on the result of the comparison:

(If Equal) Sets Flag and Exits

(If Not Equal) and Outputs the Larger to a Third Port



PROCESS: CLR F0 ;CLEAR F0 BIT (INITIALIZE)  
 IN A,P1 ;READ FIRST INPUT, STORE IN R0  
 MOV R0,A  
 IN A,P2 ;READ SECOND INPUT, STORE IN R1  
 MOV R1,A  
 CPL A ;SUBTRACT SECOND FROM FIRST  
 INC A ;2's COMPLEMENT AND ADD  
 ADD A,R0  
 JZ EQU ;BRANCH IF THEY ARE EQUAL  
 JNC SECOND ;IF NEGATIVE, SECOND WAS LARGER  
 MOV A,R0 ;ELSE, OUTPUT FIRST  
 OUTL BUS,A  
 JMP DONE ;EXIT

SECOND: MOV A,R1 ;OUTPUT SECOND  
 OUTL BUS,A  
 JMP DONE ;EXIT

EQU: CPL F0 ;SET F0  
 JMP DONE ;EXIT



## MCS-48 APPLICATION EXAMPLES

### 8 x 8 MULTIPLY-ASSEMBLED BY MCS-48 MACRO ASSEMBLER\*

ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V3.0

8 BY 8 UNSIGNED MULTIPLY

```

LOC  OBJ      LINE      SOURCE STATEMENT
=====
1= 111 $INCLUDE(:F1:MPY8)
1= 112 $INCLUDE(:F1:MPY8.PDL)
2= 113 *****
2= 114 *
2= 115 *      MPY8X8
2= 116 *
2= 117 * *****
2= 118 *
2= 119 *      THIS UTILITY PROVIDES AN 8 BY 8 UNSIGNED MULTIPLY
2= 120 *      AT ENTRY:
2= 121 *      A = LOWER EIGHT BITS OF DESTINATION OPERAND
2= 122 *      XA = DON'T CARE
2= 123 *      RI = POINTER TO SOURCE OPERAND (MULTIPLIER) IN INTERNAL MEMEORY
2= 124 *
2= 125 *      AT EXIT:
2= 126 *      A = LOWER EIGHT BITS OF RESULT
2= 127 *      XA = UPPER EIGHT BITS OF RESULT
2= 128 *      C = SET IF OVERFLOW ELSE CLEARED
2= 129 * *****
2= 130 *
2= 131 *      $MACRO ENTRYJUN=16-21 $MNOI1JUN
2= 132 *
2= 133 *      1 MPY8X8:
2= 134 *      MULTIPLICAND[15-8]=B
2= 135 *      COUNT=8
2= 136 *      REPEAT
2= 137 *      IF MULTIPLICAND[8]=B THEN BEGIN
2= 138 *      MULTIPLICAND:=MULTIPLICAND/2
2= 139 *      ELSE
2= 140 *      MULTIPLICAND[15-8]=MULTIPLICAND[15-8]+MULTIPLIER
2= 141 *      MULTIPLICAND:=MULTIPLICAND/2
2= 142 *      ENDIF
2= 143 *      COUNT:=COUNT-1
2= 144 *      UNTIL COUNT=0
2= 145 *      END MPY8X8
1= 146 ;
1= 147 ;
1= 148 $EJECT

```

\*See the application note on "Serial I/O and Math Utilities for the 8049 Microcomputer" in Intel's *Microcontroller Applications Handbook*.



## MCS-48 APPLICATION EXAMPLES

### 8 x 8 MULTIPLY-ASSEMBLED BY MCS-48 MACRO ASSEMBLER\*

ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V3.0

```

LOC  OBJ      LINE      SOURCE STATEMENT
-----
1= 149 ;1 MPY8X8:
1= 150 MPY8X8:
1= 151 ;1 MULTIPLICAND[15-0]:=0
0035 0000 1= 152      MOV      XA,000
1= 153 ;1 COUNT:=0
0037 0000 1= 154      MOV      COUNT,00
1= 155 ;1 REPEAT
1= 156 MPY8LP:
1= 157 ;2 IF MULTIPLICAND[0]=0 THEN BEGIN
0039 1243 1= 158      JBB      MPY8A
1= 159 ;3 MULTIPLICAND:=MULTIPLICAND/2
003B 2A      1= 160      XCH      A,XA
003C 97      1= 161      CLR      C
003D 67      1= 162      RRC      A
003E 2A      1= 163      XCH      A,XA
003F 67      1= 164      RRC      A
0040 EB39    1= 165      DJNZ     COUNT,MPY8LP
0042 03      1= 166      RET
1= 167 ;2 ELSE
1= 168 MPY8A:
1= 169 ;3 MULTIPLICAND[15-0]:=MULTIPLICAND[15-0]+MULTIPLIER
0043 2A      1= 170      XCH      A,XA
0044 61      1= 171      ADD      A,001
0045 67      1= 172      RRC      A
0046 2A      1= 173      XCH      A,XA
0047 67      1= 174      RRC      A
0048 EB39    1= 175      DJNZ     COUNT,MPY8LP
004A 03      1= 176      RET
1= 177 ;3 MULTIPLICAND:=MULTIPLICAND/2
1= 178 ;2 ENDIF
1= 179 ;2 COUNT:=COUNT-1
1= 180 ;1 UNTIL COUNT=0
1= 181 ;1 END MPY8X8
1= 182 $EJECT

```

\*See the application note on "Serial I/O and Math Utilities for the 8049 Microcomputer" in Intel's *Microcontroller Applications Handbook*.



## MCS-48 APPLICATION EXAMPLES

### 16 x 8 DIVIDE--(ASSEMBLED BY MCS-48 MACRO ASSEMBLER)\*

ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V3.0

| LOC  | OBJ  | LINE   | SOURCE STATEMENT                                      | OBJ | LOC |
|------|------|--------|---|-----|-----|
|      |      | 1= 183 | \$INCLUDE(:F1:DIV16)                                  |     |     |
|      |      | 1= 184 | *****   |     |     |
|      |      | 1= 185 | ;   |     |     |
|      |      | 1= 186 | ;\$DIV: DIV16   |     |     |
|      |      | 1= 187 | ;   |     |     |
|      |      | 1= 188 | *****   |     |     |
|      |      | 1= 189 | ;   |     |     |
|      |      | 1= 190 | ;   |     |     |
|      |      | 1= 191 | THIS UTILITY PROVIDES AN 16 BY 8 UNSIGNED DIVIDE      |     |     |
|      |      | 1= 192 | ;   |     |     |
|      |      | 1= 193 | AT ENTRY:   |     |     |
|      |      | 1= 194 | A = LOWER EIGHT BITS OF DESTINATION OPERAND           |     |     |
|      |      | 1= 195 | XA= UPPER EIGHT BITS OF DIVIDEND                      |     |     |
|      |      | 1= 196 | RI= POINTER TO DIVISOR IN INTERNAL MEMORY             |     |     |
|      |      | 1= 197 | ;   |     |     |
|      |      | 1= 198 | AT EXIT:  |     |     |
|      |      | 1= 199 | A = LOWER EIGHT BITS OF RESULT                        |     |     |
|      |      | 1= 200 | XA= REMAINDER   |     |     |
|      |      | 1= 201 | C = SET IF OVERFLOW ELSE CLEARED                      |     |     |
|      |      | 1= 202 | ;   |     |     |
|      |      | 1= 203 | ;   |     |     |
|      |      | 1= 204 | 1: DIV16:   |     |     |
| 004B | 2A   | 1= 205 | DIV16: XCH A,XA ; ROUTINE WORKS MOSTLY WITH BITS 15-8 |     |     |
|      |      | 1= 206 | 1: COUNT:=8   |     |     |
| 004C | 8B08 | 1= 207 | MOV COUNT,#8  |     |     |
|      |      | 1= 208 | 1: DIVIDEND[15-8]:=DIVIDEND[15-8]-DIVISOR             |     |     |
| 004E | 37   | 1= 209 | CPL A   |     |     |
| 004F | 61   | 1= 210 | ADD A,#R1   |     |     |
| 0050 | 37   | 1= 211 | CPL A   |     |     |
|      |      | 1= 212 | 1: IF BORROW=0 THEN /* IT FITS*/                      |     |     |
| 0051 | F656 | 1= 213 | JC DIVIA  |     |     |
|      |      | 1= 214 | 2: SET OVERFLOW FLAG                                  |     |     |
| 0053 | A7   | 1= 215 | CPL C   |     |     |
| 0054 | 046F | 1= 216 | JMP DIVB  |     |     |
|      |      | 1= 217 | 1: ELSE   |     |     |
|      |      | 1= 218 | DIVIA:  |     |     |
|      |      | 1= 219 | 2: RESTORE DIVIDEND                                   |     |     |
| 0056 | 61   | 1= 220 | ADD A,#R1   |     |     |
|      |      | 1= 221 | 2: REPEAT   |     |     |
|      |      | 1= 222 | DIVILP:   |     |     |
|      |      | 1= 223 | 4: DIVIDEND:=DIVIDEND*2                               |     |     |
|      |      | 1= 224 | 4: QUOTIENT:=QUOTIENT*2                               |     |     |
| 0057 | 97   | 1= 225 | CLR C   |     |     |
| 0058 | 2A   | 1= 226 | XCH A,XA  |     |     |
| 0059 | F7   | 1= 227 | RLC A   |     |     |
| 005A | 2A   | 1= 228 | XCH A,XA  |     |     |
| 005B | F7   | 1= 229 | RLC A   |     |     |
| 005C | E663 | 1= 230 | JNC DIVIE   |     |     |
| 005E | 37   | 1= 231 | CPL A   |     |     |
| 005F | 61   | 1= 232 | ADD A,#R1   |     |     |
| 0060 | 37   | 1= 233 | CPL A   |     |     |
| 0061 | 0468 | 1= 234 | JMP DIVIC   |     |     |
|      |      | 1= 235 | 4: DIVIDEND[15-8]:=DIVIDEND[15-8]-DIVISOR             |     |     |
| 0063 | 37   | 1= 236 | DIVIE: CPL A  |     |     |
| 0064 | 61   | 1= 237 | ADD A,#R1   |     |     |

\*See the application note on "Serial I/O and Math Utilities for the 8049 Microcomputer" in Intel's *Microcontroller Applications Handbook*.



## MCS-48 APPLICATION EXAMPLES

### 16 x 8 DIVIDE -(ASSEMBLED BY MCS-48 MACRO ASSEMBLER)\*

ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V3.0

| LOC  | OBJ  | LINE                          | SOURCE STATEMENT    |
|------|------|-------------------------------|---------------------|
| 0065 | 37   | 1= 238                        | CPL A               |
|      |      | 1= 239 ;                      | IF BORROW=1 THEN    |
| 0066 | E668 | 1= 240                        | JNC DIVC            |
|      |      | 1= 241 ;                      | RESTORE DIVIDEND    |
| 0068 | 61   | 1= 242                        | ADD A,0R1           |
| 0069 | 046C | 1= 243                        | JMP DIVD            |
|      |      | 1= 244 ;                      | ELSE                |
|      |      | 1= 245 DIVC: 0R1              |                     |
|      |      | 1= 246 ;                      | QUOTIENT[B]=1       |
| 006B | 1A   | 1= 247                        | INC XA              |
|      |      | 1= 248 ;                      | ENDIF               |
|      |      | 1= 249 ;                      | COUNT:=COUNT-1      |
|      |      | 1= 250 ;                      | UNTIL COUNT=0       |
| 006C | E837 | 1= 251 DIVD: DJNZ COUNT,DIVLP |                     |
|      |      | 1= 252 ;                      | CLEAR OVERFLOW FLAG |
| 006E | 97   | 1= 253                        | CLR C               |
|      |      | 1= 254 ;                      | ENDIF               |
|      |      | 1= 255 ;                      | ENDDIVIDE           |
| 006F | 2A   | 1= 256 DIVB: XCH A,XA         |                     |
| 0070 | 03   | 1= 257                        | RET                 |

\*See the application note on "Serial I/O and Math Utilities for the 8049 Microcomputer" in Intel's *Microcontroller Applications Handbook*.



---

# **MCS<sup>®</sup>-51 Architecture**

**6**

---



## CHAPTER 6

# MCS<sup>®</sup>-51 ARCHITECTURE

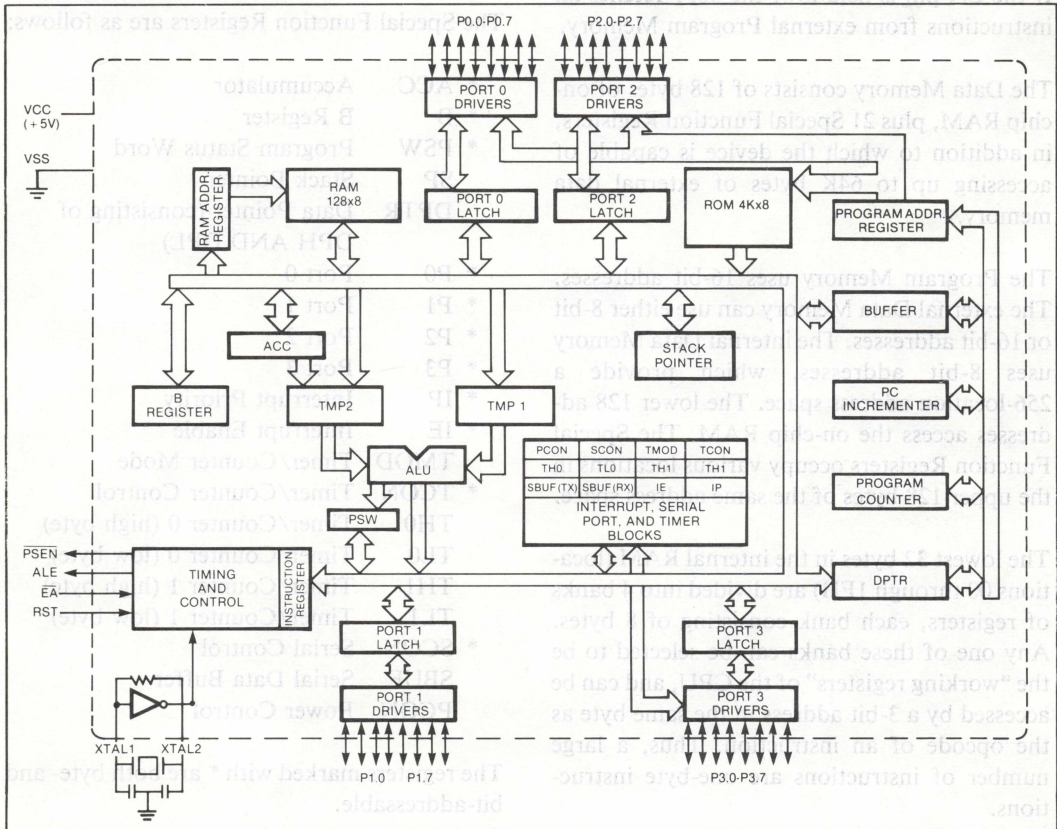
The major features of the 8051 are:

- 8-bit CPU
- on-chip oscillator
- 4K bytes of ROM
- 128 bytes of RAM
- 21 Special Function Registers
- 32 I/O lines
- 64K address space for external Data Memory
- 64K address space for external Program Memory
- two 16-bit timer/counters

- a five-source interrupt structure with two priority levels
- a full duplex serial port
- bit addressability for Boolean processing

The term "8051" is often used generically to refer to the 8051, the 8031, and the 8751. The 8031 is a ROM-less 8051; it fetches all instructions from external memory. The 8751 is an 8051 with EPROM instead of ROM.

A block diagram of the 8051 is shown in Figure 6-1. The pin-out is on the inside of the front cover of this manual.



**Figure 6-1. Block Diagram of the 8051**



## MCS-51 ARCHITECTURE

### 6.0 MEMORY ORGANIZATION

The 8051 maintains separate address spaces for Program Memory and Data Memory. The Program Memory can be up to 64K bytes long, of which the lowest 4K bytes are in the on-chip ROM.

If the  $\overline{EA}$  pin is held high, the 8051 executes out of internal ROM unless the Program Counter exceeds 0FFFH. Fetches from locations 1000H through FFFFH are directed to external Program Memory.

If the  $\overline{EA}$  pin is held low, the 8051 fetches all instructions from external Program Memory.

The Data Memory consists of 128 bytes of on-chip RAM, plus 21 Special Function Registers, in addition to which the device is capable of accessing up to 64K bytes of external data memory.

The Program Memory uses 16-bit addresses. The external Data Memory can use either 8-bit or 16-bit addresses. The internal Data Memory uses 8-bit addresses, which provide a 256-location address space. The lower 128 addresses access the on-chip RAM. The Special Function Registers occupy various locations in the upper 128 bytes of the same address space.

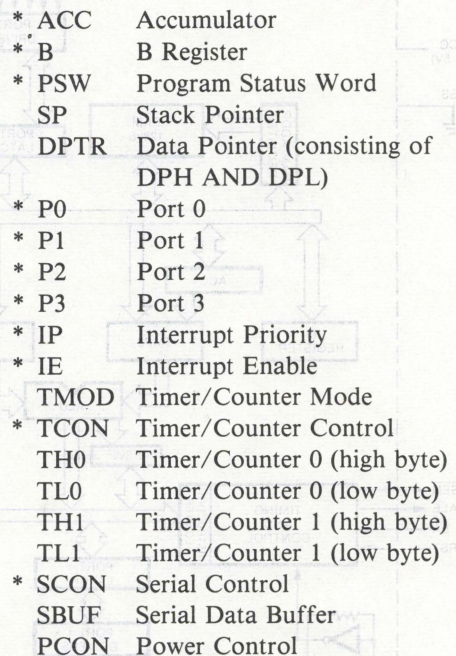
The lowest 32 bytes in the internal RAM (locations 00 through 1FH) are divided into 4 banks of registers, each bank consisting of 8 bytes. Any one of these banks can be selected to be the "working registers" of the CPU, and can be accessed by a 3-bit address in the same byte as the opcode of an instruction. Thus, a large number of instructions are one-byte instructions.

The next higher 16 bytes of the internal RAM (locations 20H through 2FH) have individually addressable bits. These are provided for use as software flags or for one-bit (Boolean) processing. This bit-addressing capability is an important feature of the 8051. In addition to the 128 individually addressable bits in RAM, eleven of the Special Function Registers also have individually addressable bits.

A memory map is shown in Figure 6-2.

### 6.1 SPECIAL FUNCTION REGISTERS

The Special Function Registers are as follows:



|        |  |
|--------|--|
| * ACC  | Accumulator                              |
| * B    | B Register                               |
| * PSW  | Program Status Word                      |
| SP     | Stack Pointer                            |
| DPTR   | Data Pointer (consisting of DPH AND DPL) |
| * P0   | Port 0                                   |
| * P1   | Port 1                                   |
| * P2   | Port 2                                   |
| * P3   | Port 3                                   |
| * IP   | Interrupt Priority                       |
| * IE   | Interrupt Enable                         |
| TMOD   | Timer/Counter Mode                       |
| * TCON | Timer/Counter Control                    |
| TH0    | Timer/Counter 0 (high byte)              |
| TL0    | Timer/Counter 0 (low byte)               |
| TH1    | Timer/Counter 1 (high byte)              |
| TL1    | Timer/Counter 1 (low byte)               |
| * SCON | Serial Control                           |
| SBUF   | Serial Data Buffer                       |
| PCON   | Power Control                            |

The registers marked with \* are both byte- and bit-addressable.



## MCS-51 ARCHITECTURE

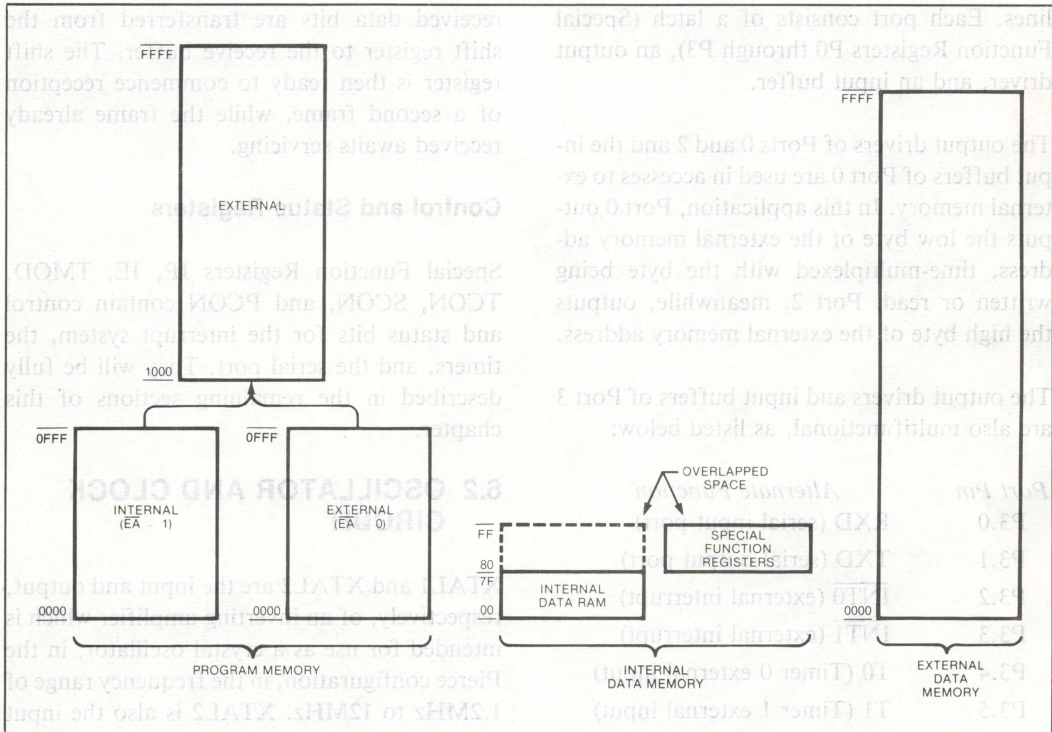


Figure 6-2. 8051 Memory Map

### Accumulator

ACC is the Accumulator. The mnemonics for accumulator-specific instructions refer to the accumulator simply as A, but the register itself is named ACC.

### B Register

The B register is used during multiply and divide operations. For other instructions it can be treated as another scratch register.

### Stack Pointer

The Stack Pointer is 8 bits wide. The stack can

reside anywhere in the 128 bytes of on-chip RAM. When the 8051 is reset, the stack pointer is initialized to 07H. When executing a PUSH or a CALL, the stack pointer is incremented before data is stored, so the stack would begin at location 08H.

### Data Pointer

The Data Pointer (DPTR) is a 16-bit register, consisting of a high byte (DPH) and a low byte (DPL). Its intended function is to hold a 16-bit address.

### Ports 0 through 3

These four parallel ports provide the 32 I/O



## MCS-51 ARCHITECTURE

lines. Each port consists of a latch (Special Function Registers P0 through P3), an output driver, and an input buffer.

The output drivers of Ports 0 and 2 and the input buffers of Port 0 are used in accesses to external memory. In this application, Port 0 outputs the low byte of the external memory address, time-multiplexed with the byte being written or read. Port 2, meanwhile, outputs the high byte of the external memory address.

The output drivers and input buffers of Port 3 are also multifunctional, as listed below:

| Port Pin | Alternate Function   |
|----------|--|
| P3.0     | RXD (serial input port)                                    |
| P3.1     | TXD (serial output port)                                   |
| P3.2     | $\overline{\text{INT0}}$ (external interrupt)              |
| P3.3     | $\overline{\text{INT1}}$ (external interrupt)              |
| P3.4     | T0 (Timer 0 external input)                                |
| P3.5     | T1 (Timer 1 external input)                                |
| P3.6     | $\overline{\text{WR}}$ (external Data Memory write strobe) |
| P3.7     | $\overline{\text{RD}}$ (external Data Memory read strobe)  |

### Serial Data Buffer

The Serial Buffer is actually two separate registers. When data is moved to SBUF, it goes to the transmit buffer where it is held for serial transmission. (Moving a byte to SBUF is what initiates the transmission.) When data is moved from SBUF, it comes from the receive buffer.

During serial reception the incoming bits are clocked into a separate shift register. When reception of a frame is complete, and if various other conditions are satisfied, 8

received data bits are transferred from the shift register to the receive buffer. The shift register is then ready to commence reception of a second frame, while the frame already received awaits servicing.

### Control and Status Registers

Special Function Registers IP, IE, TMOD, TCON, SCON, and PCON contain control and status bits for the interrupt system, the timers, and the serial port. They will be fully described in the remaining sections of this chapter.

## 6.2 OSCILLATOR AND CLOCK CIRCUIT

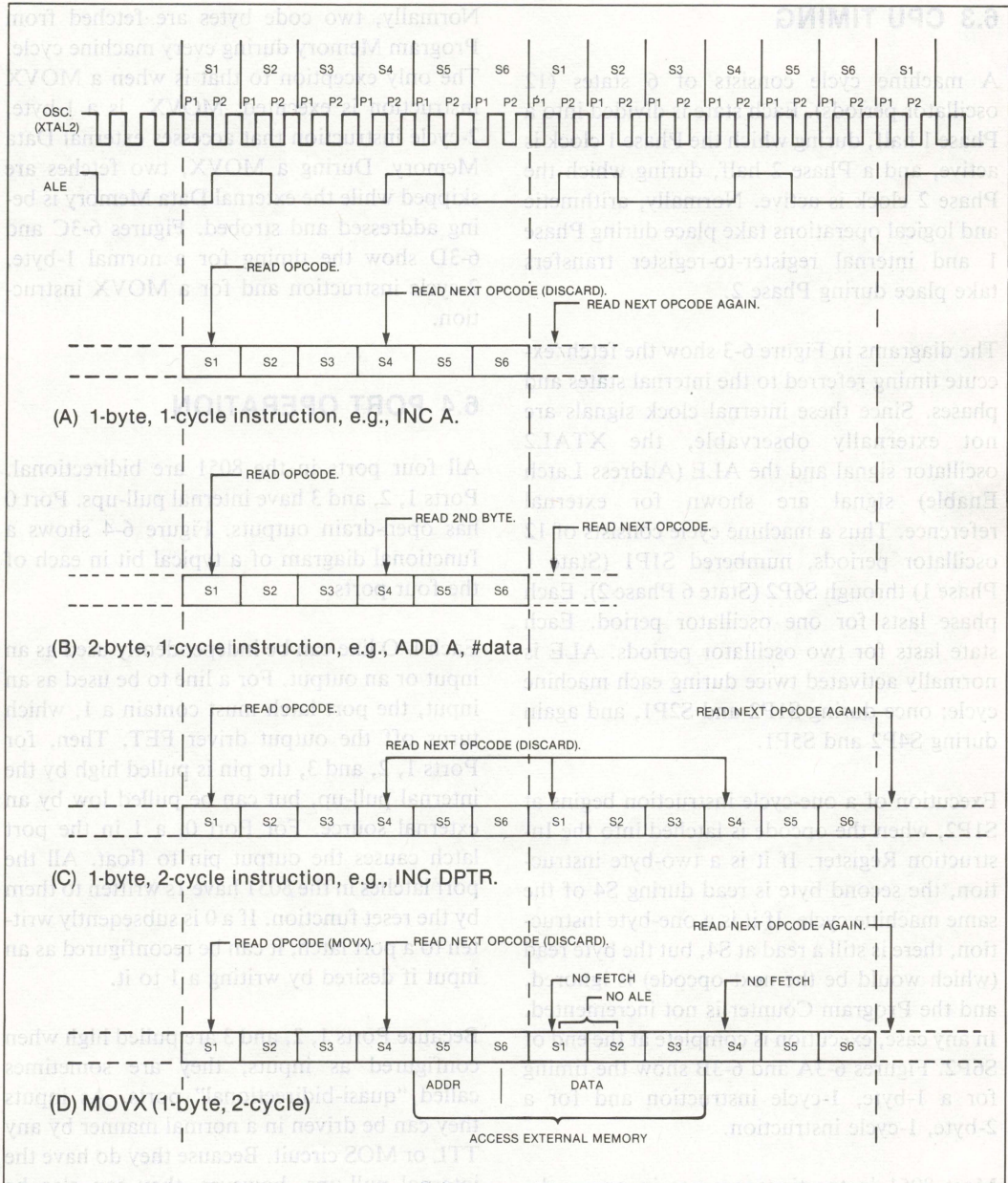
XTAL1 and XTAL2 are the input and output, respectively, of an inverting amplifier which is intended for use as a crystal oscillator, in the Pierce configuration, in the frequency range of 1.2MHz to 12MHz. XTAL2 is also the input to the internal clock generator.

To drive the chip with an external oscillator, one would ground XTAL1 and drive XTAL2. Since the input to the clock generator is a divide-by-two flip-flop, there are no requirements on the duty cycle of the external oscillator signal. However, minimum high and low times must be observed.

The clock generator divides the oscillator frequency by 2, and provides a two-phase clock signal to the chip. The Phase 1 signal is active during the first half of each clock period, and the Phase 2 signal is active during the second half of each clock period.



## MCS-51 ARCHITECTURE



**Figure 6-3. Fetch/Execute Sequences in the 8051**



## 6.3 CPU TIMING

A machine cycle consists of 6 states (12 oscillator periods). Each state is divided into a Phase 1 half, during which the Phase 1 clock is active, and a Phase 2 half, during which the Phase 2 clock is active. Normally, arithmetic and logical operations take place during Phase 1 and internal register-to-register transfers take place during Phase 2.

The diagrams in Figure 6-3 show the fetch/execute timing referred to the internal states and phases. Since these internal clock signals are not externally observable, the XTAL2 oscillator signal and the ALE (Address Latch Enable) signal are shown for external reference. Thus a machine cycle consists of 12 oscillator periods, numbered S1P1 (State 1 Phase 1) through S6P2 (State 6 Phase 2). Each phase lasts for one oscillator period. Each state lasts for two oscillator periods. ALE is normally activated twice during each machine cycle: once during S1P2 and S2P1, and again during S4P2 and S5P1.

Execution of a one-cycle instruction begins at S1P2, when the opcode is latched into the Instruction Register. If it is a two-byte instruction, the second byte is read during S4 of the same machine cycle. If it is a one-byte instruction, there is still a read at S4, but the byte read (which would be the next opcode) is ignored, and the Program Counter is not incremented. In any case, execution is complete at the end of S6P2. Figures 6-3A and 6-3B show the timing for a 1-byte, 1-cycle instruction and for a 2-byte, 1-cycle instruction.

Most 8051 instructions execute in one cycle. MUL (multiply) and DIV (divide) are the only instructions that take more than two cycles to complete. They take four cycles.

Normally, two code bytes are fetched from Program Memory during every machine cycle. The only exception to that is when a MOVX instruction is executed. MOVX is a 1-byte, 2-cycle instruction that accesses external Data Memory. During a MOVX, two fetches are skipped while the external Data Memory is being addressed and strobed. Figures 6-3C and 6-3D show the timing for a normal 1-byte, 2-cycle instruction and for a MOVX instruction.

## 6.4 PORT OPERATION

All four ports in the 8051 are bidirectional. Ports 1, 2, and 3 have internal pull-ups. Port 0 has open-drain outputs. Figure 6-4 shows a functional diagram of a typical bit in each of the four ports.

Each I/O line can be independently used as an input or an output. For a line to be used as an input, the port latch must contain a 1, which turns off the output driver FET. Then, for Ports 1, 2, and 3, the pin is pulled high by the internal pull-up, but can be pulled low by an external source. For Port 0, a 1 in the port latch causes the output pin to float. All the port latches in the 8051 have 1s written to them by the reset function. If a 0 is subsequently written to a port latch, it can be reconfigured as an input if desired by writing a 1 to it.

Because Ports 1, 2, and 3 are pulled high when configured as inputs, they are sometimes called "quasi-bidirectional" ports. As inputs they can be driven in a normal manner by any TTL or MOS circuit. Because they do have the internal pull-ups, however, they can also be driven by open-collector or open-drain outputs without the need for additional external pull-ups.



## MCS-51 ARCHITECTURE

Port 0 differs in not having internal pull-ups. The upper FET in the P0 output driver (see Figure 6-4A) is turned off except when the port is being used as an ADDR/DATA bus in accesses to external memory. Consequently, P0 lines that are being used as output ports have open-drain outputs. Writing a 1 to a P0 latch results in both output FETs being turned off, so the pin floats. In that condition it can be used as a high-impedance input.

Notice in Figure 6-4 that there are two ways to read a port: an instruction reads either the latch or the pin. In the 8051, some instructions read the latch and some read the pin. The instructions that read the latch rather than the

pin are the ones that read a value, possibly change it, and then rewrite it to the latch. These are called “read-modify-write” instructions. The instructions listed below are read-modify-write instructions. When the destination operand is a port or a port bit, these instructions read the latch rather than the pin:

ANL (logical AND, e.g., ANL P1, A)  
 ORL (logical OR, e.g., ORL P2, A)  
 XRL (logical EX-OR, e.g., XRL P3, A)

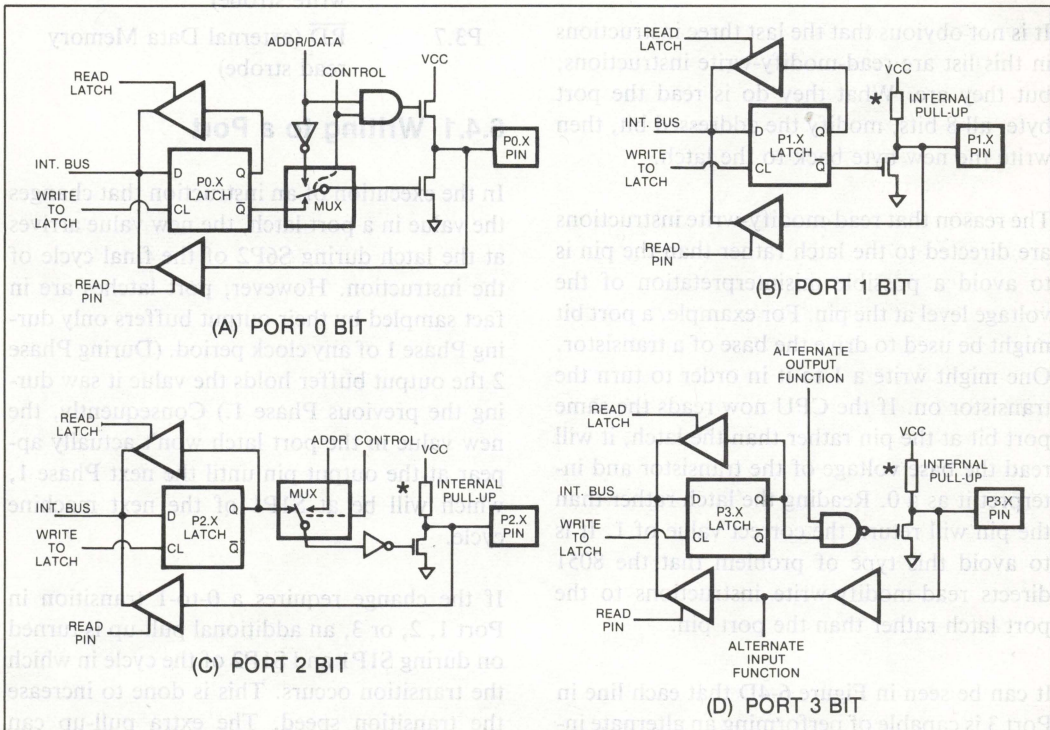


Figure 6-4. Port Latches and Buffers (\*See Figure 6-5 for details of internal pullup).



## MCS-51 ARCHITECTURE

|            |  |
|------------|--|
| JBC        | (jump if bit = 1 and clear bit, e.g., JBC P1.1, LABEL) |
| CPL        | (complement bit, e.g., CPL P3.0)                       |
| INC        | (increment, e.g., INC P2)                              |
| DEC        | (decrement, e.g., DEC P2)                              |
| DJNZ       | (decrement and jump if not zero, e.g., DJNZ P3, LABEL) |
| MOV PX.Y,C | (move carry bit to bit Y of Port X)                    |
| CLR PX.Y   | (clear bit Y of Port X)                                |
| SET PX.Y   | (set bit Y of Port X)                                  |

It is not obvious that the last three instructions in this list are read-modify-write instructions, but they are. What they do is read the port byte, all 8 bits, modify the addressed bit, then write the new byte back to the latch.

The reason that read-modify-write instructions are directed to the latch rather than the pin is to avoid a possible misinterpretation of the voltage level at the pin. For example, a port bit might be used to drive the base of a transistor. One might write a 1 to it in order to turn the transistor on. If the CPU now reads the same port bit at the pin rather than the latch, it will read the base voltage of the transistor and interpret it as a 0. Reading the latch rather than the pin will return the correct value of 1. It is to avoid this type of problem that the 8051 directs read-modify-write instructions to the port latch rather than the port pin.

It can be seen in Figure 6-4D that each line in Port 3 is capable of performing an alternate input function and an alternate output function, not related to the port function. The bit latch

must contain a 1, or else the port pin will be stuck at 0 regardless of which alternate function is trying to do what with it. The alternate functions that are implemented have already been listed, but the list is repeated here for convenience:

| Port Pin | Alternate Function   |
|----------|--|
| P3.0     | RXD (serial input port)                                    |
| P3.1     | TXD (serial output port)                                   |
| P3.2     | $\overline{\text{INT0}}$ (external interrupt)              |
| P3.3     | $\overline{\text{INT1}}$ (external interrupt)              |
| P3.4     | T0 (Timer 0 external input)                                |
| P3.5     | T1 (Timer 1 external input)                                |
| P3.6     | $\overline{\text{WR}}$ (external Data Memory write strobe) |
| P3.7     | $\overline{\text{RD}}$ (external Data Memory read strobe)  |

### 6.4.1 Writing to a Port

In the execution of an instruction that changes the value in a port latch, the new value arrives at the latch during S6P2 of the final cycle of the instruction. However, port latches are in fact sampled by their output buffers only during Phase 1 of any clock period. (During Phase 2 the output buffer holds the value it saw during the previous Phase 1.) Consequently, the new value in the port latch won't actually appear at the output pin until the next Phase 1, which will be at S1P1 of the next machine cycle.

If the change requires a 0-to-1 transition in Port 1, 2, or 3, an additional pull-up is turned on during S1P1 and S1P2 of the cycle in which the transition occurs. This is done to increase the transition speed. The extra pull-up can source about 100 times the current that the normal pull-up can.



## MCS-51 ARCHITECTURE

It should be noted that the internal pull-ups are field-effect transistors, not linear resistors. The pull-up arrangement is shown in Figure 6-5. The fixed part of the pull-up is a depletion-mode transistor with the gate wired to the source. If the port pin is shorted to ground, this transistor will allow about 0.25 mA (typical) to exit the pin. In parallel with the fixed pull-up is an enhancement-mode transistor which is activated during S1 whenever the port bit does a 0-to-1 transition. During this interval, if the port pin is shorted to ground, this extra transistor will allow an additional 30 mA (typical) to exit the pin.

### 6.4.2 Port Loading

The output buffers of Ports 1, 2, and 3 can drive 3 LS TTL inputs. The output buffers of Port 0 can each drive 8 LS TTL inputs.

Ports 1, 2, and 3 can drive any MOS input without the need for external pull-ups. Port 0 needs external pull-ups to drive MOS inputs, except when it's being used as an AD-

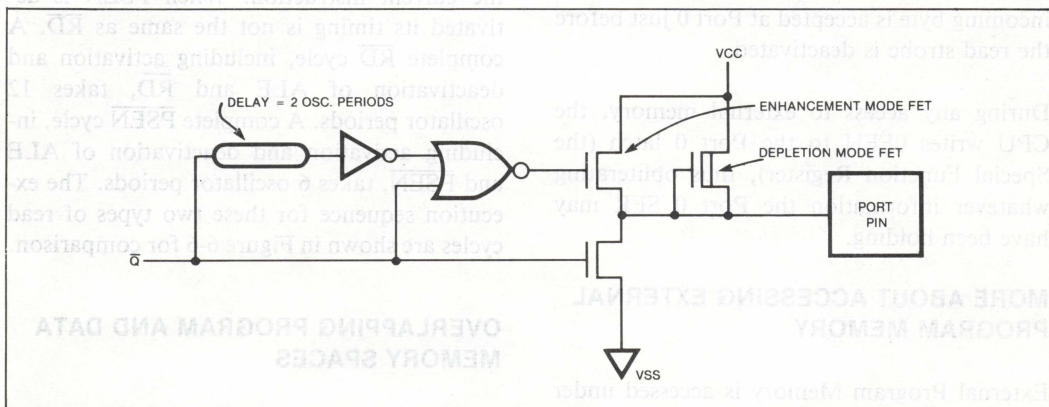
DRESS/DATA bus, in which case it can drive MOS inputs without external pull-ups.

## 6.5 ACCESSING EXTERNAL MEMORY

Accesses to external memory are of two types: accesses to external Program Memory and accesses to external Data Memory. Accesses to external Program Memory use signal  $\overline{\text{PSEN}}$  (program store enable) as the read strobe. Accesses to external Data Memory use  $\overline{\text{RD}}$  or  $\overline{\text{WR}}$  (alternate functions of P3.7 and P3.6) to strobe the memory.

Fetches from external Program Memory always use a 16-bit address. Accesses to external Data Memory can use either a 16-bit address ( $\text{MOVX @DPTR}$ ) or an 8-bit address ( $\text{MOVX @Ri}$ ).

Whenever a 16-bit address is used, the high byte of the address comes out on Port 2, where it is held for the duration of the read or write cycle. During this time the Port 2 latch (the



**Figure 6-5. Internal Pull-Ups.** The enhancement mode transistor is turned on for 2 osc. periods whenever Q makes a 1-to-0 transition.



## MCS-51 ARCHITECTURE

Special Function Register) does not have to contain 1s, and the contents of the Port 2 SFR are not modified. If the external memory cycle is not immediately followed by another external memory cycle, the undisturbed contents of the Port 2 SFR will reappear shortly after the read or write strobe is deactivated.

If an 8-bit address is being used ( $\text{MOVX @Ri}$ ), the contents of the Port 2 SFR remain at the Port 2 pins throughout the external memory cycle. This will facilitate paging.

In any case, the low byte of the address is time-multiplexed with the data byte on Port 0. The  $\text{ADDR/DATA}$  signal drives both FETs in the Port 0 output buffers. Thus in this application the Port 0 pins are not open-drain outputs, and they don't need external pull-ups. Signal ALE (address latch enable) should be used to capture the address byte into an external latch. The address byte is valid at the negative transition of ALE. Then, in a write cycle, the data byte to be written appears on Port 0 just before  $\overline{\text{WR}}$  is activated, and remains there until after  $\overline{\text{WR}}$  is deactivated. In a read cycle, the incoming byte is accepted at Port 0 just before the read strobe is deactivated.

During any access to external memory, the CPU writes 0FFH to the Port 0 latch (the Special Function Register), thus obliterating whatever information the Port 0 SFR may have been holding.

### MORE ABOUT ACCESSING EXTERNAL PROGRAM MEMORY

External Program Memory is accessed under two conditions:

- 1) Whenever the program counter (PC) contains a number that is larger than 0FFFH;

- 2) Whenever signal  $\overline{\text{EA}}$  is active, regardless of the contents of PC.

The 8031 is an 8051 that doesn't have internal Program Memory.  $\overline{\text{EA}}$  must be externally wired to low on the 8031 to enable it to fetch the lower 4K program bytes from external memory.

When the CPU is executing out of external Program Memory, all 8 bits of Port 2 are dedicated to an output function: During external program fetches they output the high byte of the PC, and during accesses to external Data Memory they output either DPH or the Port 2 SFR (depending on whether the external Data Memory access is a  $\text{MOVX @DPTR}$  or a  $\text{MOVX @Ri}$ ).

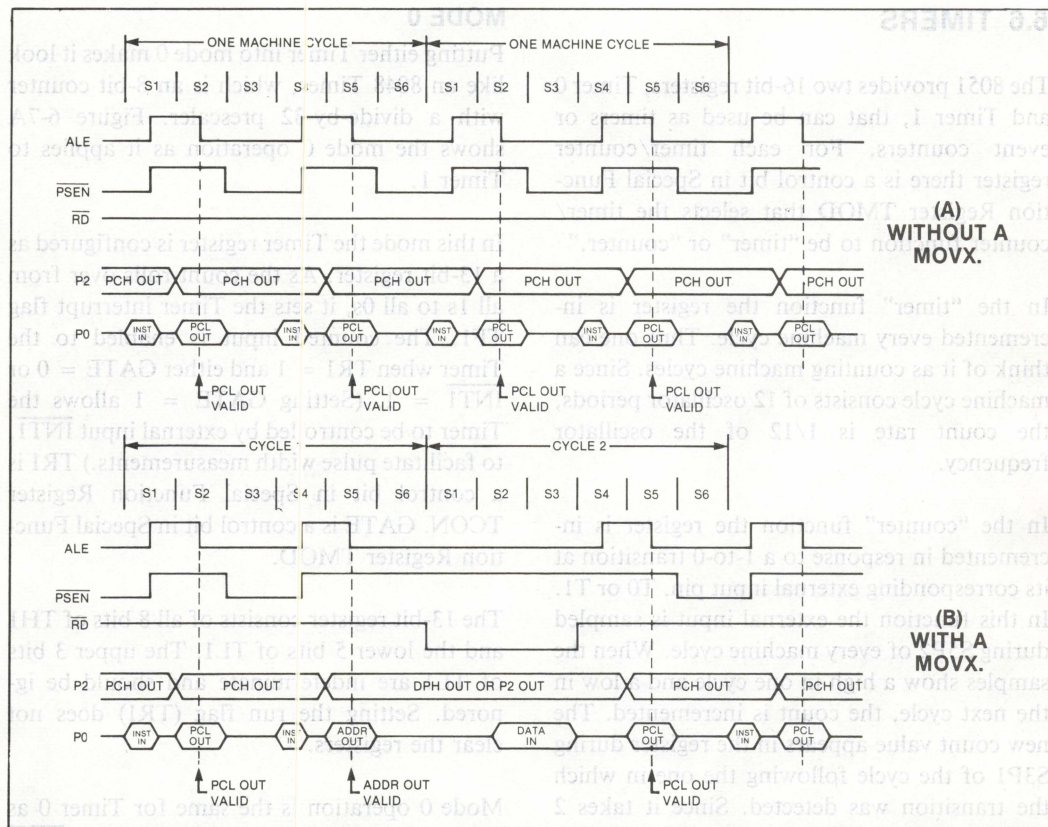
The read strobe for external fetches is  $\overline{\text{PSEN}}$ .  $\overline{\text{PSEN}}$  is not activated for internal fetches. When the CPU is accessing external Program Memory,  $\overline{\text{PSEN}}$  is activated twice every cycle (except during a  $\text{MOVX}$  instruction) whether or not the byte fetched is actually needed for the current instruction. When  $\overline{\text{PSEN}}$  is activated its timing is not the same as  $\overline{\text{RD}}$ . A complete  $\overline{\text{RD}}$  cycle, including activation and deactivation of ALE and  $\overline{\text{RD}}$ , takes 12 oscillator periods. A complete  $\overline{\text{PSEN}}$  cycle, including activation and deactivation of ALE and  $\overline{\text{PSEN}}$ , takes 6 oscillator periods. The execution sequence for these two types of read cycles are shown in Figure 6-6 for comparison.

### OVERLAPPING PROGRAM AND DATA MEMORY SPACES

In some applications it is desirable to execute a program from the same physical memory that is being used to store data. In the 8051 the external Program and Data Memory spaces can



# MCS-51 ARCHITECTURE



**Figure 6-6. Executing Out of External Program Memory**  
(Detailed timing diagrams are in the data sheets.)

be combined by ANDing  $\overline{\text{PSEN}}$  and  $\overline{\text{RD}}$ . A positive-logic AND of these two signals produces an active-low read strobe that can be used for the combined physical memory. Since the  $\overline{\text{PSEN}}$  cycle is faster than the  $\overline{\text{RD}}$  cycle, the external memory needs to be fast enough to accommodate the  $\overline{\text{PSEN}}$  cycle.

## MORE ABOUT ALE

The main function of ALE is to provide a properly timed signal to latch the low byte of an address from P0 to an external latch during

fetches from external Program Memory. For that purpose ALE is activated twice every machine cycle. This activation takes place even when the cycle involves no external fetch. The only time an ALE pulse doesn't come out is during an access to external Data Memory. The first ALE of the second cycle of a MOVX instruction is missing (see Figure 6-6). Consequently, in any system that does not use external Data Memory, ALE is activated at a constant rate of 1/6 the oscillator frequency, and can be used for external clocking or timing purposes.



## 6.6 TIMERS

The 8051 provides two 16-bit registers, Timer 0 and Timer 1, that can be used as timers or event counters. For each timer/counter register there is a control bit in Special Function Register TMOD that selects the timer/counter function to be “timer” or “counter.”

In the “timer” function the register is incremented every machine cycle. Thus one can think of it as counting machine cycles. Since a machine cycle consists of 12 oscillator periods, the count rate is 1/12 of the oscillator frequency.

In the “counter” function the register is incremented in response to a 1-to-0 transition at its corresponding external input pin, T0 or T1. In this function the external input is sampled during S5P2 of every machine cycle. When the samples show a high in one cycle and a low in the next cycle, the count is incremented. The new count value appears in the register during S3P1 of the cycle following the one in which the transition was detected. Since it takes 2 machine cycles (24 oscillator periods) to recognize a 1-to-0 transition, the maximum count rate is 1/24 of the oscillator frequency. There are no restrictions on the duty cycle of the external input signal, but to ensure that a given level is sampled at least once before it changes, it should hold for at least one full machine cycle.

In addition to the “timer” or “counter” selection, each timer/counter has four operating modes from which to select. These modes are functionally illustrated in Figure 6-7. Operating modes 0, 1, and 2 are the same for both timer/counters. Mode 3 is different. The four operating modes are described below.

### MODE 0

Putting either Timer into mode 0 makes it look like an 8048 Timer, which is an 8-bit counter with a divide-by-32 prescaler. Figure 6-7A shows the mode 0 operation as it applies to Timer 1.

In this mode the Timer register is configured as a 13-bit register. As the count rolls over from all 1s to all 0s, it sets the Timer interrupt flag TF1. The counted input is enabled to the Timer when TR1 = 1 and either GATE = 0 or INT1 = 1. (Setting GATE = 1 allows the Timer to be controlled by external input INT1, to facilitate pulse width measurements.) TR1 is a control bit in Special Function Register TCON. GATE is a control bit in Special Function Register TMOD.

The 13-bit register consists of all 8 bits of TH1 and the lower 5 bits of TL1. The upper 3 bits of TL1 are indeterminate and should be ignored. Setting the run flag (TR1) does not clear the registers.

Mode 0 operation is the same for Timer 0 as for Timer 1. Substitute TR0, TF0, and INT0 for the corresponding Timer 1 signals in Figure 6-7A. There are two different GATE bits, one for Timer 1 (TMOD.7) and one for Timer 0 (TMOD.3).

### MODE 1

Mode 1 is the same as Mode 0, except that the Timer register is being run with all 16 bits.

### MODE 2

Mode 2 configures the Timer register as an 8-bit counter (TL1) with automatic reload. Overflow from TL1 not only sets TF1 but also reloads TL1 with the contents of TH1, which is preset by software to any desired one-byte



## MCS-51 ARCHITECTURE

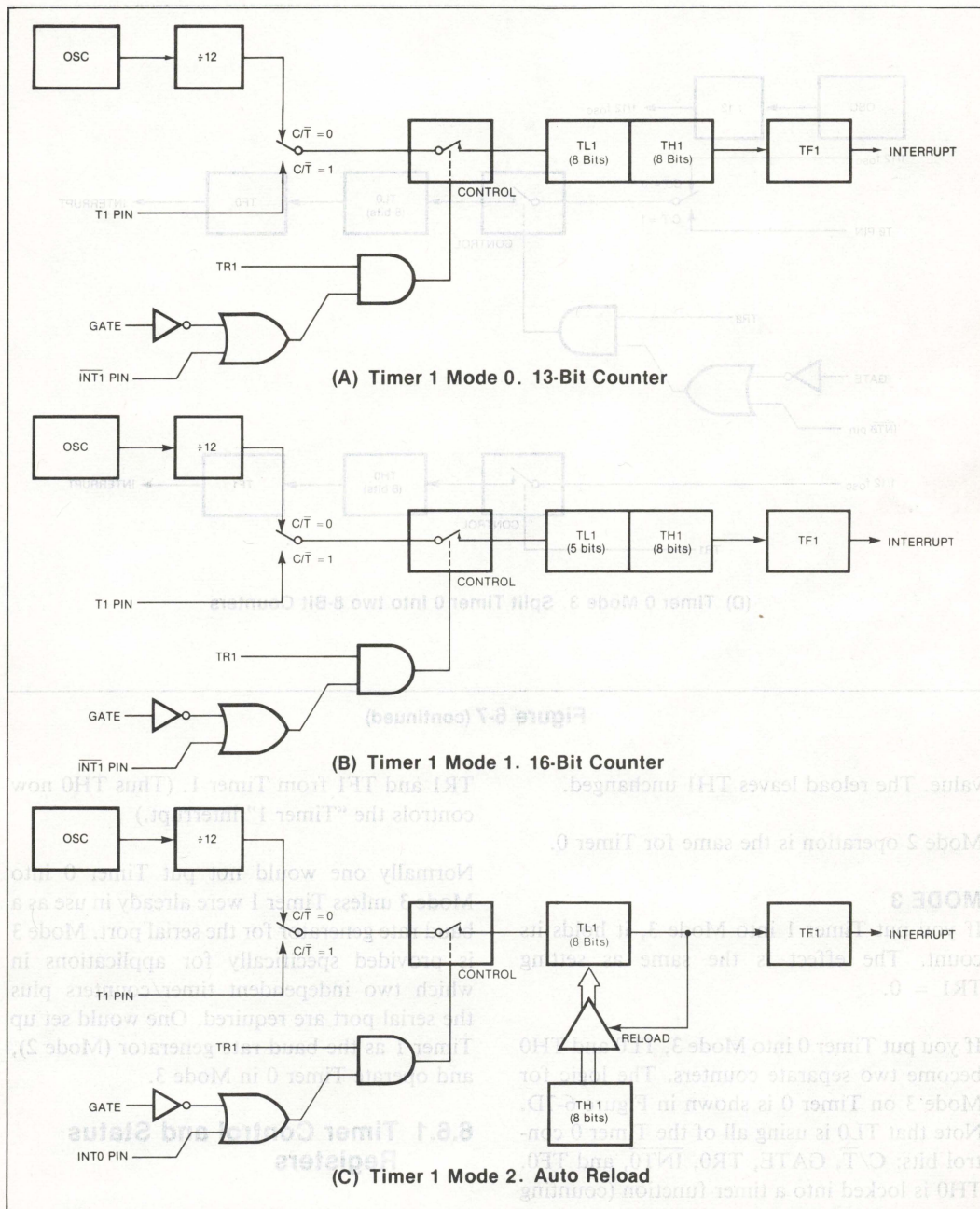


Figure 6-7



## MCS-51 ARCHITECTURE

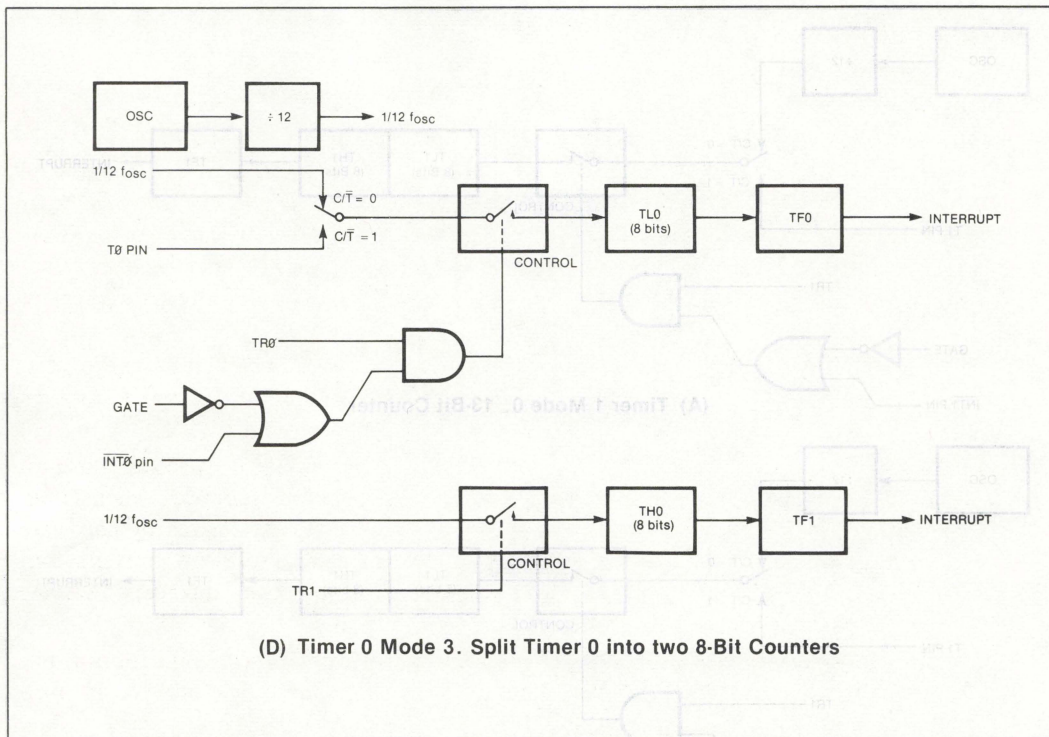


Figure 6-7 (continued)

value. The reload leaves TH1 unchanged.

Mode 2 operation is the same for Timer 0.

### MODE 3

If you put Timer 1 into Mode 3, it holds its count. The effect is the same as setting  $TR1 = 0$ .

If you put Timer 0 into Mode 3, TL0 and TH0 become two separate counters. The logic for Mode 3 on Timer 0 is shown in Figure 6-7D. Note that TL0 is using all of the Timer 0 control bits:  $C/\overline{T}$ , GATE, TR0,  $\overline{INT0}$ , and TF0. TH0 is locked into a timer function (counting machine cycles) and has taken over the use of

TR1 and TF1 from Timer 1. (Thus TH0 now controls the "Timer 1" interrupt.)

Normally one would not put Timer 0 into Mode 3 unless Timer 1 were already in use as a baud rate generator for the serial port. Mode 3 is provided specifically for applications in which two independent timer/counters plus the serial port are required. One would set up Timer 1 as the baud rate generator (Mode 2), and operate Timer 0 in Mode 3.

### 6.6.1 Timer Control and Status Registers

Two Special Function Registers, TMOD and



## MCS-51 ARCHITECTURE

TCON, are used to define the operating modes and control the functions of the timer/counters. When an instruction changes the content of TMOD or TCON, the change is latched into the SFR and takes effect at S1P1 of the next instruction's first cycle. The registers are shown below.

### TMOD: Timer Mode Control Register

Bit: 7 6 5 4 3 2 1 0  
 GATE C/T M1 M0 GATE C/T M1 M0  
 /----- Timer 1 -----/ /----- Timer 0 -----/

where M1, M0 specify the Mode, as follows:

| M1 | M0 | Mode | Description   |
|----|----|------|---|
| 0  | 0  | 0    | 13-bit counter  |
| 0  | 1  | 1    | 16-bit counter  |
| 1  | 0  | 2    | 8-bit counter with auto reload                        |
| 1  | 1  | 3    | split Timer 0 into two 8-bit counters or stop Timer 1 |

- C/T selects "counter" or "timer" function. Set for "counter" function (count negative transitions at T0 or T1 pin). Clear for "timer" function (count machine cycles).
- GATE is Gating Control. When set, Timer "x" is enabled only while  $\overline{\text{INTx}}$  pin is high and TRx bit is set. When cleared, Timer "x" is enabled whenever TRx bit is set.

**Note:** All bits of TMOD are cleared by reset.

### TCON: Timer Control Register

Bit: 7 6 5 4 3 2 1 0  
 TF1 TR1 TF0 TR0 IE1 IT1 IE0 IT0

where

- TF1 is the Timer 1 overflow interrupt flag. Set by hardware when Timer 1 overflows. Cleared by hardware when the processor transfers control to the interrupt service routine.
- TR1 is the Timer 1 run control bit. Set/cleared by software to turn Timer 1 on/off.
- TF0 is the Timer 0 overflow interrupt flag. Set by hardware when Timer 0 overflows. Cleared by hardware when the processor transfers control to the interrupt service routine.
- TR0 is the Timer 0 run control bit. Set/cleared by software to turn Timer 0 on/off.
- IE1 is the external interrupt 1 edge flag. If IT1 = 1, this bit is set by hardware when  $\overline{\text{INT1}}$  is detected to have made a 1-to-0 transition. Cleared by hardware when the processor transfers control to the interrupt service routine.
- IT1 determines whether external interrupt 1 is edge-triggered or level-triggered. If IT1 = 1, external interrupt 1 is edge-triggered. If IT1 = 0, external interrupt is triggered by a detected low at  $\overline{\text{INT1}}$  rather than a 1 in IE1.
- IE0 is the external interrupt 0 edge flag. If IT0 = 1, this bit is set by hardware when  $\overline{\text{INT0}}$  is detected to



## MCS-51 ARCHITECTURE

have made a 1-to-0 transition. Cleared by hardware when the processor transfers control to the interrupt service routine.

- **IT0** determines whether external interrupt 0 is edge-triggered or level-triggered. If  $IT0 = 1$ , external interrupt 0 is edge-triggered. If  $IT0 = 0$ , external interrupt 0 is triggered by a detected low at  $\overline{INT0}$  rather than a 1 in  $IE0$ .

Bits  $IE1$  through  $IT0$  have to do with the external interrupts, and are more fully discussed in the section dealing with the interrupt structure.

*Note:* All bits of  $TCON$  are cleared by reset.

### 6.7 SERIAL INTERFACE

The serial port is full duplex, meaning it can transmit and receive simultaneously. It is also receive-buffered, meaning it can commence reception of a second byte before a previously received byte has been read from the receive register. (However, if the first byte still hasn't been read by the time reception of the second byte is complete, one of the bytes will be lost.) The serial port registers are both accessed at Special Function Register  $SBUF$ . A write to  $SBUF$  loads the transmit register, and a read accesses a physically separate receive register.

The serial port can operate in 4 modes:

**Mode 0:** Serial data enters and exits through  $RXD$ .  $TXD$  outputs the shift clock. 8 bits are transmitted/received: 8 data bits (LSB first). The baud rate is fixed at  $1/12$  the oscillator frequency.

**Mode 1:** 10 bits are transmitted (through  $TXD$ ) or received (through  $RXD$ ): a start bit

(0), 8 data bits (LSB first), and a stop bit (1). On receive, the stop bit goes into  $RB8$  in Special Function Register  $SCON$ . The baud rate is variable.

**Mode 2:** 11 bits are transmitted (through  $TXD$ ) or received (through  $RXD$ ): a start bit (0), 8 data bits (LSB first), a programmable 9th data bit, and a stop bit (1). On transmit, the 9th data bit ( $TB8$ ) can be assigned the value of 0 or 1. With nominal software overhead,  $TB8$  can be made a parity bit, as shown in Figure 6-8. On receive, the 9th data bit goes into  $RB8$  in Special Function Register  $SCON$ , and the stop bit is ignored. The baud rate is programmable to either  $1/32$  or  $1/64$  the oscillator frequency.

```
MOV C,P      ; Parity moved to carry (byte  
              already in A)  
MOV TB8, C   ; Put carry into Transmit  
              Bit 8  
MOV SBUF, A  ; Load Transmit Register
```

**Figure 6-8. Generating Parity and Initiating a Transmission**

**Mode 3:** 11 bits are transmitted (through  $TXD$ ) or received (through  $RXD$ ): a start bit (0), 8 data bits (LSB first), a programmable 9th data bit, and a stop bit (1). In fact, Mode 3 is the same as Mode 2 in all respects except the baud rate. The baud rate in Mode 3 is variable.

Modes 2 and 3 have a special provision for multiprocessor communications. In these modes 9 data bits are received. The 9th one goes into  $RB8$ . Then comes a stop bit. The port can be programmed such that when the stop bit is received, the serial port interrupt will be activated only if  $RB8 = 1$ . This feature is enabled by setting bit  $SM2$  in  $SCON$ . The way to use this feature in multiprocessor systems is as follows.



When the master processor wants to transmit a block of data to one of several slaves, it first sends out an address byte which identifies the target slave. An address byte differs from a data byte in that the 9th bit is 1 in an address byte and 0 in a data byte. With SM2 = 1, no slave will be interrupted by a data byte. An address byte, however, will interrupt all slaves, so that each slave can examine the received byte and see if it is being addressed. The addressed slave will clear its SM2 bit and prepare to receive the data bytes that will be coming. The slaves that weren't addressed leave their SM2s set and go on about their business, ignoring the coming data bytes.

SM2 should be cleared for operation in Mode 0 or 1.

## 6.7.1 Baud Rates

The baud rate in Mode 0 is fixed at 1/12 the oscillator frequency. The baud rate in Mode 2 is either 1/64 or 1/32 of the oscillator frequency, depending on the value of bit SMOD in Special Function Register PCON. If SMOD = 0 (which is its value on reset), the baud rate is 1/64 the oscillator frequency. If SMOD = 1, the baud rate is 1/32 of the oscillator frequency.

Baud rates in Modes 1 and 3 are determined by the Timer 1 overflow rate, as shown below:

$$\text{Baud Rate} = (\text{Timer 1 overflow rate}) / n$$

where n is an integer whose value is either 32 or 16, depending on the value of bit SMOD in Special Function Register PCON. If SMOD = 0 (which is its value on reset), n = 32. If SMOD = 1, n = 16. Timer 1 can be configured in any mode. The Timer 1 overflow

rate is determined by its count rate and how many counts it takes to reach overflow.

For example, Timer 1 can be configured in the auto reload mode (TMOD.5 = 1, TMOD.4 = 0). The Timer must be running (TCON.6 = 1), and to keep the overflows from generating unnecessary interrupts the Timer 1 interrupt should be disabled (IE.3 = 0). Then the overflow rate depends on the reload value in TH1, as follows:

$$\text{Overflow Rate} = (\text{count rate}) / [256 - (\text{TH1})]$$

For very low baud rates one might select the 16-bit Timer 1 mode (TMOD.5 = 0, TMOD.4 = 1), and use the Timer 1 interrupt to do a software reload. In this case one would want to have the Timer 1 interrupt enabled (IE.3 = 1).

In any case, if Timer 1 is running with bit C/T = 0, the count rate is 1/12 the oscillator frequency. If the timer is running with C/T = 1, the count rate is the external input frequency, whose maximum usable value is 1/24 the oscillator frequency.

Figure 6-9 lists various commonly used baud rates and how they can be obtained in the 8051.

|                  |                  |      | TIMER 1 |      |              |
|------------------|------------------|------|---------|------|--------------|
| BAUD RATE        | f <sub>osc</sub> | SMOD | C/T     | MODE | RELOAD VALUE |
| MODE 0 MAX: 1MHZ | 12 MHZ           | X    | X       | X    | X            |
| MODE 2 MAX: 375K | 12 MHZ           | 1    | X       | X    | X            |
| MODES 1,3: 62.5K | 12 MHZ           | 1    | 0       | 2    | FFH          |
| 19.2K            | 11.059 MHZ       | 1    | 0       | 2    | FDH          |
| 9.6K             | 11.059 MHZ       | 0    | 0       | 2    | FDH          |
| 4.8K             | 11.059 MHZ       | 0    | 0       | 2    | FAH          |
| 2.4K             | 11.059 MHZ       | 0    | 0       | 2    | F4H          |
| 1.2K             | 11.059 MHZ       | 0    | 0       | 2    | E8H          |
| 137.5            | 11.986 MHZ       | 0    | 0       | 2    | 1DH          |
| 110              | 6MHZ             | 0    | 0       | 2    | 72H          |
| 110              | 12 MHZ           | 0    | 0       | 1    | FEEDH        |

**Figure 6-9. Commonly Used Baud Rates**



## 6.7.2 Baud Rate Bit in PCON

PCON is a Special Function Register (address = 87H) which has been added to the 8051 to implement certain Power Control options in the CMOS version of the chip. In the HMOS chip all the bits in PCON are dummy except bit 7, which is SMOD. SMOD is used in both the HMOS and CMOS versions to double the baud rate in modes 1, 2, and 3. PCON is not bit-addressable.

The reset value of SMOD is 0. Writing a 1 to SMOD (MOV PCON, #80H or MOV 87H, #80H) doubles the baud rate in modes 1, 2, and 3.

## 6.7.3 Serial Port Control Register

Special Function Register SCON is used to define the operating mode and control certain functions of the serial port. It also receives the 9th data bit (RB8), and contains the transmit and receive interrupt flags (TI and RI). The register is as shown below:

### SCON: Serial Port Control Register

Bit: 7 6 5 4 3 2 1 0  
SM0 SM1 SM2 REN TB8 RB8 TI RI

where SM0, SM1 specify the serial port mode, as follows:

| SM0 | SM1 | Mode | Description    | Baud Rate                    |
|-----|-----|------|----------------|------------------------------|
| 0   | 0   | 0    | shift register | $f_{osc}/12$                 |
| 0   | 1   | 1    | 8-bit UART     | variable                     |
| 1   | 0   | 2    | 9-bit UART     | $f_{osc}/64$ or $f_{osc}/32$ |
| 1   | 1   | 3    | 9-bit UART     | variable                     |

- SM2 enables the multiprocessor communication feature in modes 2 and 3. In mode 2 or 3, if SM2 is set to 1 then RI will not be activated if the received 9th data bit (RB8) is 0. In mode 1, if SM2 = 1 then RI will not be activated if a valid stop bit was not received. In mode 0, SM2 should be 0.
- REN enables serial reception. Set by software to enable reception. Clear by software to disable reception.
- TB8 is the 9th data bit that will be transmitted in modes 2 and 3. Set or clear by software as desired.
- RB8 in modes 2 and 3, is the 9th data bit that was received. In mode 1, if SM2 = 0, RB8 is the stop bit that was received. In mode 0, RB8 is not used.
- TI is transmit interrupt flag. Set by hardware at the end of the 8th bit time in mode 0, or at the beginning of the stop bit in the other modes, in any serial transmission. Must be cleared by software.
- RI is receive interrupt flag. Set by hardware at the end of the 8th bit time in mode 0, or halfway through the stop bit time in the other modes, in any serial reception (except see SM2). Must be cleared by software.

**Note:** All bits of SCON are cleared by reset.



## MCS-51 ARCHITECTURE

When an instruction changes the content of SCON, the change is latched into the SFR and takes effect at S1P1 of the next instruction's first cycle. However, if a serial transmission is already under way, the TB8 that will go out is the old value, not the new value.

**Serial Port Interrupts:** When transmission of a serial frame is complete, flag bit TI is activated, which in turn activates the serial port interrupt. The same serial port interrupt is activated by flag bit RI when an incoming frame has been received. Thus the CPU normally enters the serial port interrupt service routine without knowing beforehand whether the interrupt was generated by TI or RI. Both flags are located in the Special Function Register SCON. Neither flag is cleared by hardware. The interrupt service routine must clear the flag that generated the interrupt, or else another interrupt will be generated by the same flag.

### 6.7.4 Serial Port Data Registers

In all serial modes a write to SBUF loads the same 9-bit shift register. The data byte goes into the first 8 bits, with the LSB at the output bit of the register. The write to SBUF also loads the 9th bit of the shift register with either a 1 or TB8, depending on the mode. And it initiates the transmission.

The receive registers are an input shift register which is 8 bits wide in mode 0 and 9 bits wide in the other modes, plus SBUF itself, a read-only register which is loaded by the hardware with the data byte at the same time that RI is activated. In the UART modes, the 9th bit is loaded into RB8 in SCON at the same time

that the data byte is loaded into SBUF. RB8 and SBUF are not changed if SM2 causes the received data to be ignored.

### 6.7.5 More About Mode 0

Serial data enters and exits through RXD. TXD outputs the shift clock. 8 bits are transmitted/received: 8 data bits (LSB first). The baud rate is fixed at 1/12 the oscillator frequency.

Figure 6-10 shows a somewhat simplified functional diagram of the serial port in mode 0, and associated timing diagrams.

Transmission is initiated by any instruction that uses SBUF as a destination register. The "write to SBUF" signal at S6P2 also loads a 1 into the 9th bit position of the transmit shift register and tells the TX Control block to commence a transmission. The internal timing is such that one full machine cycle will elapse between "write to SBUF" and activation of SEND.

SEND enables the output of the shift register to the alternate output function line of P3.0, and also enables SHIFT CLOCK to the alternate output function line of P3.1. SHIFT CLOCK is low during S3, S4, and S5 of every machine cycle, and high during S6, S1, and S2. At S6P2 of every machine cycle in which SEND is active, the contents of the transmit shift register are shifted to the right one position.

As data bits shift out to the right, zeroes come in from the left. When the MSB of the data byte is at the output position of the shift



# MCS-51 ARCHITECTURE

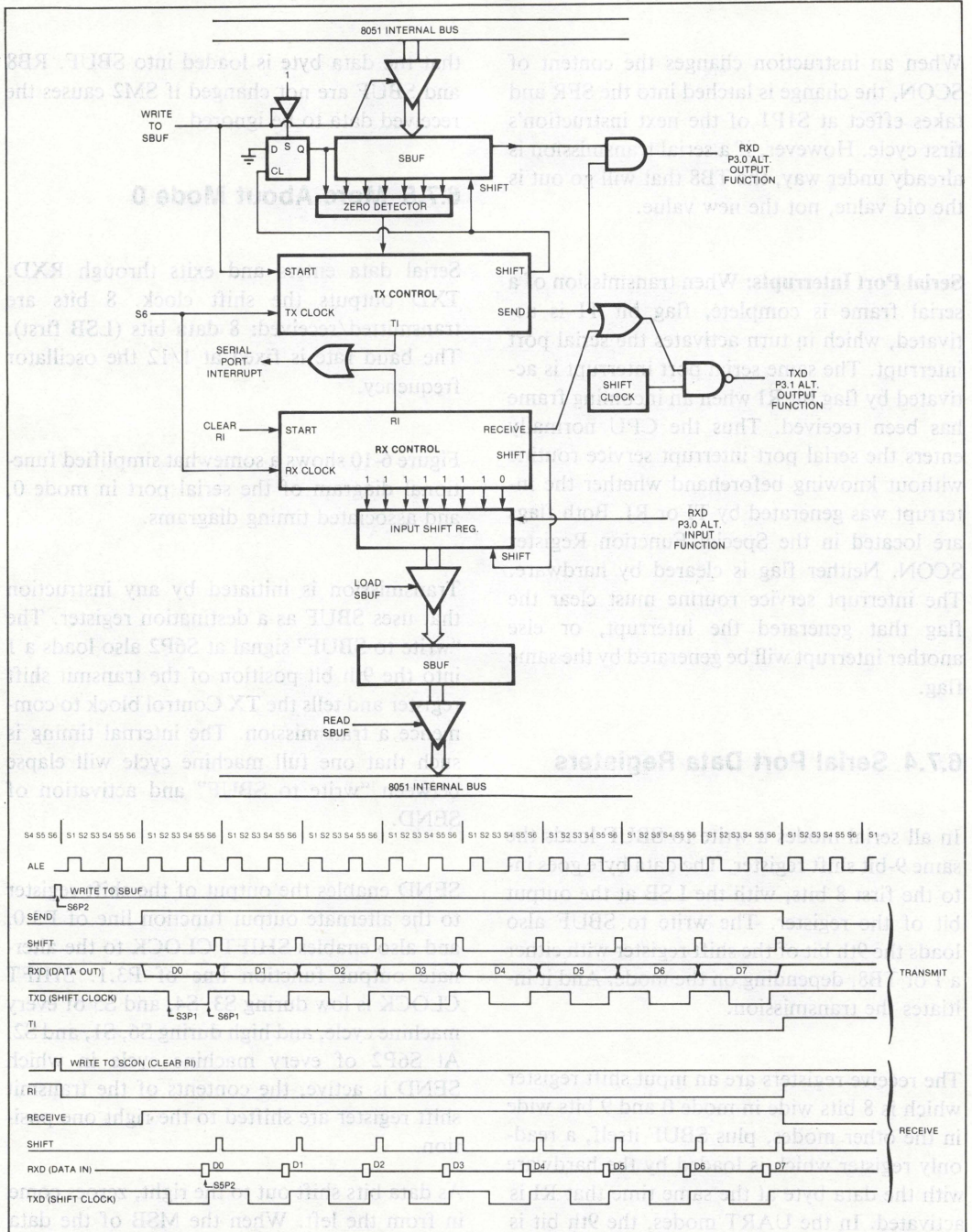


Figure 6-10. Serial Port in Mode 0



## MCS-51 ARCHITECTURE

register, then the 1 that was initially loaded into the 9th position is just to the left of the MSB, and all positions to the left of that contain zeroes. This condition flags the TX Control block to do one last shift and then deactivate SEND and set TI. Both of these actions occur at S1P1 of the 10th machine cycle after “write to SBUF.”

Reception is initiated by clearing RI, provided REN = 1. At S6P2 of the next machine cycle the RX Control unit writes the bits 11111110 to the receive shift register, and in the next clock phase activates RECEIVE.

RECEIVE enables SHIFT CLOCK to the alternate output function line of P3.1. SHIFT CLOCK makes transitions at S3P1 and S6P1 of every machine cycle. At S6P2 of every machine cycle in which RECEIVE is active, the contents of the receive shift register are shifted to the left one position. The value that comes in from the right is the value that was sampled at the P3.0 pin at S5P2 of the same machine cycle.

As data bits come in from the right, 1s shift out to the left. When the 0 that was initially loaded into the rightmost position arrives at the leftmost position in the shift register, it flags the RX Control block to do one last shift and load SBUF. At S1P1 of the 10th machine cycle after the write to SCON that cleared RI, RECEIVE is cleared and RI is set.

**Mode 0 Applications:** Mode 0 was intended primarily for I/O expansion using CMOS or TTL shift registers, as shown in Figure 6-11.

### 6.7.6 More About Mode 1

Ten bits are transmitted (through TXD) or received (through RXD): a start bit (0), 8 data

bits (LSB first), and a stop bit (1). On receive, the stop bit goes into RB8 in SCON. The baud rate is determined by the Timer 1 overflow rate.

Figure 6-12 shows a somewhat simplified functional diagram of the serial port in Mode 1, and associated timing diagrams for transmit and receive.

Transmission is initiated by any instruction that uses SBUF as a destination register. The “write to SBUF” signal also loads a 1 into the 9th bit position of the transmit shift register and flags the TX Control unit that a transmission is requested. Transmission actually commences at S1P1 of the machine cycle following the next rollover in the divide-by-16 counter. (Thus the bit times are synchronized to the divide-by-16 counter, not to the “write to SBUF” signal.)

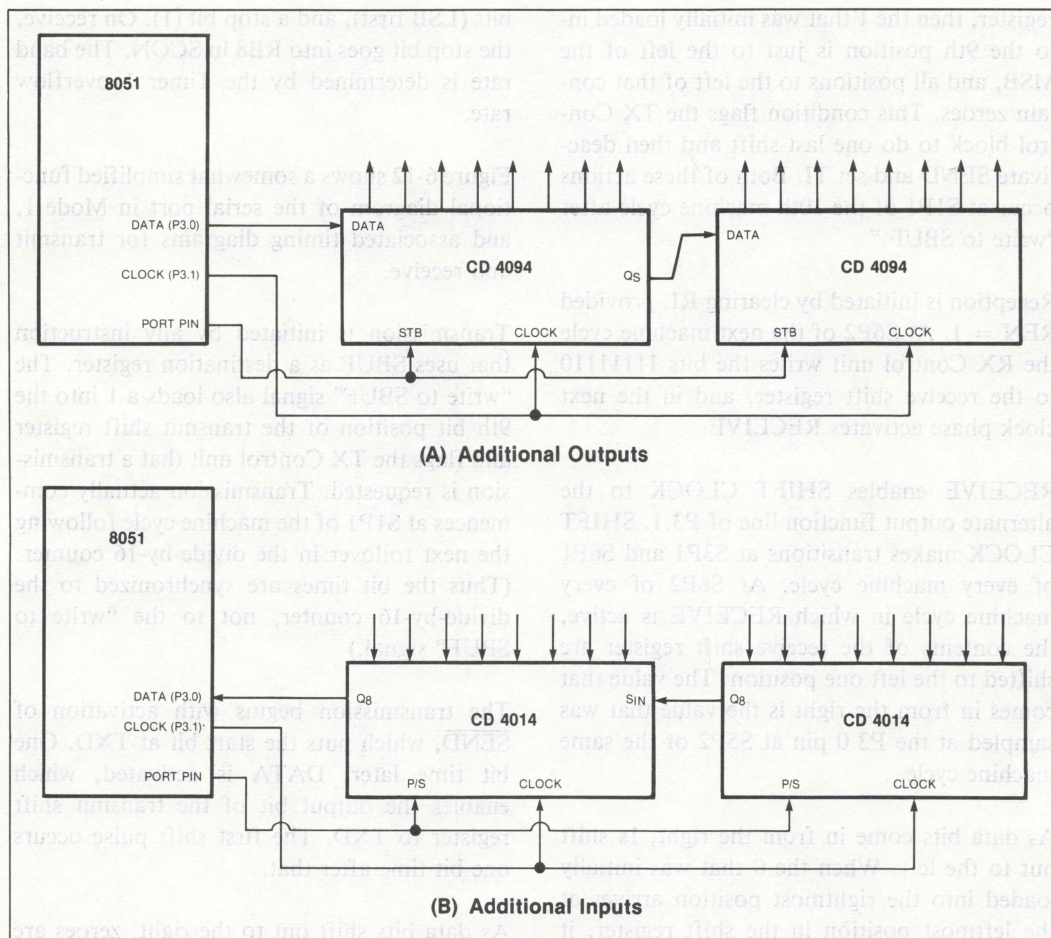
The transmission begins with activation of SEND, which puts the start bit at TXD. One bit time later, DATA is activated, which enables the output bit of the transmit shift register to TXD. The first shift pulse occurs one bit time after that.

As data bits shift out to the right, zeroes are clocked in from the left. When the MSB of the data byte is at the output position of the shift register, then the 1 that was initially loaded into the 9th position is just to the left of the MSB, and all positions to the left of that contain zeroes. This condition flags the TX Control unit to do one last shift and then deactivate SEND and set TI. This occurs at the 10th divide-by-16 rollover after “write to SBUF.”

Reception is initiated by a detected 1-to-0 transition at RXD. For this purpose RXD is sam-



## MCS-51 ARCHITECTURE



**Figure 6-11. Mode 0 Applications**



## MCS-51 ARCHITECTURE

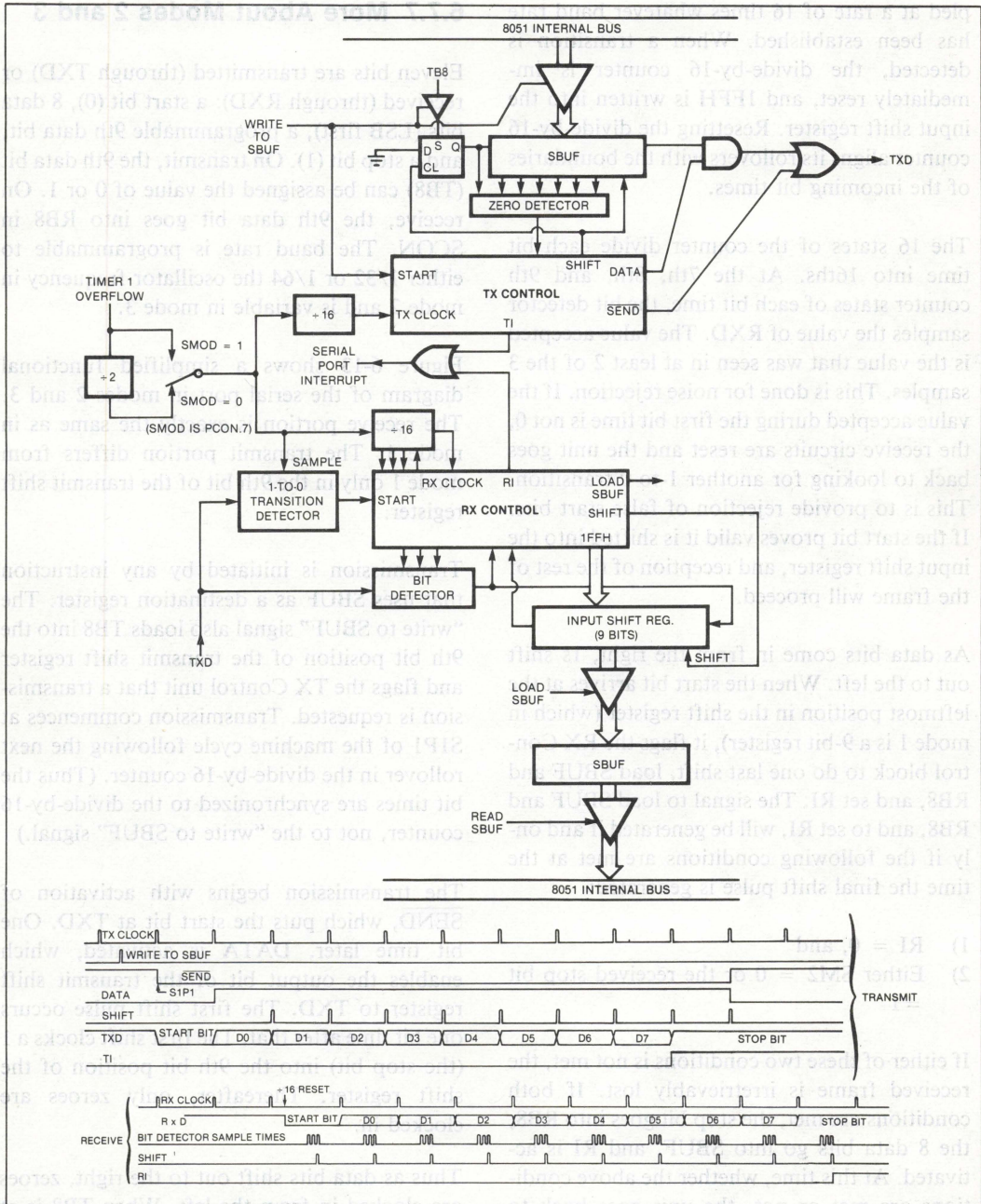


Figure 6-12. Serial Port in Mode 1



pled at a rate of 16 times whatever baud rate has been established. When a transition is detected, the divide-by-16 counter is immediately reset, and 1FFH is written into the input shift register. Resetting the divide-by-16 counter aligns its rollovers with the boundaries of the incoming bit times.

The 16 states of the counter divide each bit time into 16ths. At the 7th, 8th, and 9th counter states of each bit time, the bit detector samples the value of RXD. The value accepted is the value that was seen in at least 2 of the 3 samples. This is done for noise rejection. If the value accepted during the first bit time is not 0, the receive circuits are reset and the unit goes back to looking for another 1-to-0 transition. This is to provide rejection of false start bits. If the start bit proves valid it is shifted into the input shift register, and reception of the rest of the frame will proceed.

As data bits come in from the right, 1s shift out to the left. When the start bit arrives at the leftmost position in the shift register (which in mode 1 is a 9-bit register), it flags the RX Control block to do one last shift, load SBUF and RB8, and set RI. The signal to load SBUF and RB8, and to set RI, will be generated if and only if the following conditions are met at the time the final shift pulse is generated:

- 1) RI = 0, and
- 2) Either SM2 = 0 or the received stop bit = 1

If either of these two conditions is not met, the received frame is irretrievably lost. If both conditions are met, the stop bit goes into RB8, the 8 data bits go into SBUF, and RI is activated. At this time, whether the above conditions are met or not, the unit goes back to looking for a 1-to-0 transition in RXD.

### 6.7.7 More About Modes 2 and 3

Eleven bits are transmitted (through TXD) or received (through RXD): a start bit (0), 8 data bits (LSB first), a programmable 9th data bit, and a stop bit (1). On transmit, the 9th data bit (TB8) can be assigned the value of 0 or 1. On receive, the 9th data bit goes into RB8 in SCON. The baud rate is programmable to either 1/32 or 1/64 the oscillator frequency in mode 2 and is variable in mode 3.

Figure 6-13 shows a simplified functional diagram of the serial port in modes 2 and 3. The receive portion is exactly the same as in mode 1. The transmit portion differs from mode 1 only in the 9th bit of the transmit shift register.

Transmission is initiated by any instruction that uses SBUF as a destination register. The "write to SBUF" signal also loads TB8 into the 9th bit position of the transmit shift register and flags the TX Control unit that a transmission is requested. Transmission commences at S1P1 of the machine cycle following the next rollover in the divide-by-16 counter. (Thus the bit times are synchronized to the divide-by-16 counter, not to the "write to SBUF" signal.)

The transmission begins with activation of SEND, which puts the start bit at TXD. One bit time later, DATA is activated, which enables the output bit of the transmit shift register to TXD. The first shift pulse occurs one bit time after that. The first shift clocks a 1 (the stop bit) into the 9th bit position of the shift register. Thereafter, only zeroes are clocked in.

Thus as data bits shift out to the right, zeroes are clocked in from the left. When TB8 is at the output position of the shift register, then



## MCS-51 ARCHITECTURE

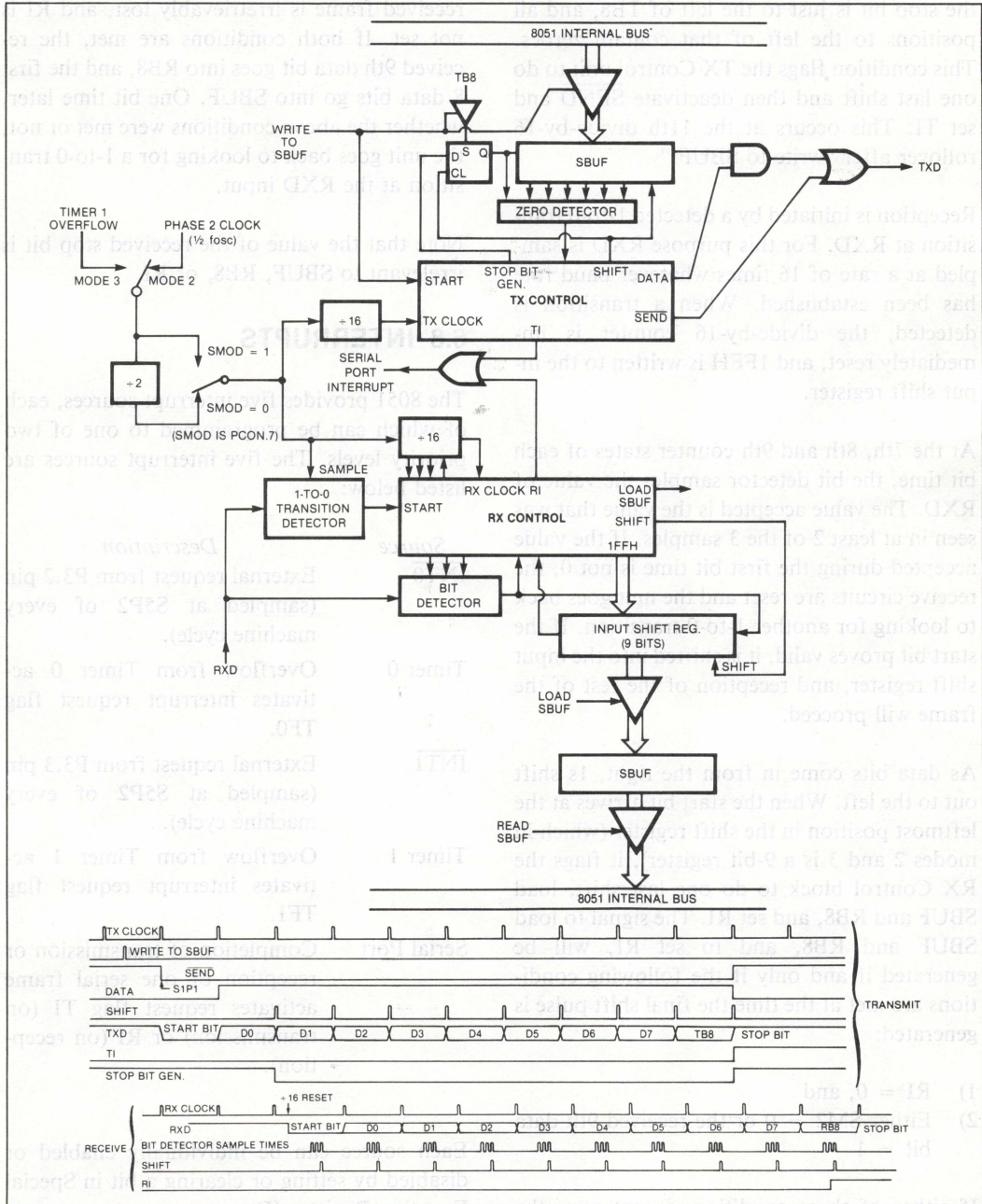


Figure 6-13. Serial Port in Modes 2 and 3



## MCS-51 ARCHITECTURE

the stop bit is just to the left of TB8, and all positions to the left of that contain zeroes. This condition flags the TX Control unit to do one last shift and then deactivate  $\overline{\text{SEND}}$  and set TI. This occurs at the 11th divide-by-16 rollover after "write to SBUF."

Reception is initiated by a detected 1-to-0 transition at RXD. For this purpose RXD is sampled at a rate of 16 times whatever baud rate has been established. When a transition is detected, the divide-by-16 counter is immediately reset, and 1FFH is written to the input shift register.

At the 7th, 8th and 9th counter states of each bit time, the bit detector samples the value of RXD. The value accepted is the value that was seen in at least 2 of the 3 samples. If the value accepted during the first bit time is not 0, the receive circuits are reset and the unit goes back to looking for another 1-to-0 transition. If the start bit proves valid, it is shifted into the input shift register, and reception of the rest of the frame will proceed.

As data bits come in from the right, 1s shift out to the left. When the start bit arrives at the leftmost position in the shift register (which in modes 2 and 3 is a 9-bit register), it flags the RX Control block to do one last shift, load SBUF and RB8, and set RI. The signal to load SBUF and RB8, and to set RI, will be generated if and only if the following conditions are met at the time the final shift pulse is generated:

- 1) RI = 0, and
- 2) Either SM2 = 0 or the received 9th data bit = 1

If either of these conditions is not met, the

received frame is irretrievably lost, and RI is not set. If both conditions are met, the received 9th data bit goes into RB8, and the first 8 data bits go into SBUF. One bit time later, whether the above conditions were met or not, the unit goes back to looking for a 1-to-0 transition at the RXD input.

Note that the value of the received stop bit is irrelevant to SBUF, RB8, or RI.

## 6.8 INTERRUPTS

The 8051 provides five interrupt sources, each of which can be programmed to one of two priority levels. The five interrupt sources are listed below:

| Source                   | Description   |
|--------------------------|---|
| $\overline{\text{INT0}}$ | External request from P3.2 pin (sampled at S5P2 of every machine cycle).  |
| Timer 0                  | Overflow from Timer 0 activates interrupt request flag TF0.   |
| $\overline{\text{INT1}}$ | External request from P3.3 pin (sampled at S5P2 of every machine cycle).  |
| Timer 1                  | Overflow from Timer 1 activates interrupt request flag TF1.   |
| Serial Port              | Completion of transmission or reception of one serial frame activates request flag TI (on transmission) or RI (on reception). |

Each source can be individually enabled or disabled by setting or clearing a bit in Special Function Register IE.



## MCS-51 ARCHITECTURE

Each source can be programmed to a high-priority level or a low-priority level by setting or clearing a bit in Register IP. A low-priority interrupt can itself be interrupted by a high-priority interrupt, but not by another low-priority interrupt. A high-priority interrupt can't be interrupted. To implement these rules, the Interrupt System contains two non-addressable "priority level active" flip-flops. One indicates that a high-priority interrupt is being serviced, and blocks all further interrupts. The other indicates that a low-priority interrupt is being serviced, and blocks all but high-priority interrupts.

In the event that requests of the same priority level are received simultaneously, an internal polling sequence determines which request is serviced. Thus within each priority level there is a second priority structure determined by the polling sequence, as follows:

| Source               | Priority within Level |
|----------------------|-----------------------|
| External Interrupt 0 | (highest)             |
| Timer 0 Overflow     |                       |
| External Interrupt 1 |                       |
| Timer 1 Overflow     |                       |
| Serial Port          | (lowest)              |

All the interrupt sources are examined sequentially during each cycle, such that by S6 of any cycle all active interrupt requests have been found and prioritized. Response to the active request of highest priority will commence with state 1 of the next machine cycle, provided the response is not blocked by any of the following conditions:

- 1) An interrupt of equal or higher priority level is already in progress.
- 2) The current machine cycle is not the final

cycle in the execution of the instruction in progress. (In other words, no interrupt request will be responded to until the instruction in progress is completed.)

- 3) The instruction in progress is RETI or an access to Special Function Registers IE or IP. (In other words an interrupt request will not be responded to after RETI or after a read or write to IE or IP until at least one other instruction has been executed.)

If any of the above conditions exists, the result of the interrupt poll is discarded. If none of the above conditions exists, the result of the interrupt poll is acted on with the very next machine cycle.

The processor acknowledges a request by first setting the appropriate "priority level active" flip-flop. Then it executes a hardware subroutine call to the servicing routine. It also clears the flag that requested the interrupt (with exceptions: it doesn't clear  $\overline{\text{INT0}}$  or  $\overline{\text{INT1}}$ , since it has no control over the sources of these signals, and it doesn't clear TI or RI). The hardware subroutine call pushes the contents of the Program Counter onto the stack (but it does not save the PSW) and reloads the PC with an address that depends on the source of the interrupt request, as shown below:

| Source               | Address |
|----------------------|---------|
| External Interrupt 0 | 0003H   |
| Timer 0 Overflow     | 000BH   |
| External Interrupt 1 | 0013H   |
| Timer 1 Overflow     | 001BH   |
| Serial Port          | 0023H   |

Execution proceeds from that address until the RETI instruction is encountered.

The RETI instruction clears the "priority level



active" flip-flop that was set when this interrupt was acknowledged. Then it pops the top two bytes from the stack and reloads the Program Counter. Execution of the interrupted program commences from where it left off.

### 6.8.1 More About External Interrupts

The external sources can be programmed to be level-activated or transition-activated by setting or clearing bit IT1 or IT0 in Register TCON. If  $ITx = 0$ , external interrupt  $x$  is triggered by a detected low at the  $\overline{INTx}$  pin. If  $ITx = 1$ , external interrupt  $x$  is edge-triggered. In this mode if successive samples of the  $\overline{INTx}$  pin show a high in one cycle and a low in the next cycle, interrupt request flag IEx in TCON is set. Flag bit IEx then requests the interrupt.

Since the external interrupt pins are sampled once each machine cycle, an input high or low should hold for at least 12 oscillator periods to ensure sampling. If the external interrupt is transition-activated, the external source has to hold the request pin high for at least one cycle, and then hold it low for at least one cycle to ensure that the transition is seen so that interrupt request flag IEx will be set. IEx will be automatically cleared by the CPU when the service routine is called.

If the external interrupt is level-activated, the external source has to hold the request active until the requested interrupt is actually generated. Then it has to deactivate the request before the interrupt service routine is completed, or else another interrupt will be generated.

### 6.8.2 Response Time to External Interrupt

The  $\overline{INT0}$  and  $\overline{INT1}$  levels are latched into an internal holding register at S5P2 of every machine cycle. The values are not actually polled by the circuitry until the next machine cycle. If a request is active and conditions are right for it to be acknowledged, a hardware subroutine call to the requested service routine will be the next instruction to be executed. The call itself takes two cycles. Thus a minimum of three complete machine cycles elapse between activation of an external interrupt request and the beginning of execution of the first instruction of the service routine.

A longer response time would result if the request is blocked by one of the 3 previously listed conditions. If an interrupt of equal or higher priority level is already in progress, the additional wait time obviously depends on the nature of the other interrupt's service routine. If the instruction in progress is not in its final cycle, the additional wait time can't be more than 3 cycles, since the longest instructions (MUL and DIV) are only 4 cycles long. If the instruction in progress is RETI or an access to IE or IP, the additional wait time can't be more than 5 cycles (a maximum of one more cycle to complete the instruction in progress plus 4 cycles to complete the next instruction if the next instruction is MUL or DIV).

Thus, in a single-interrupt system, the response time is always more than 3 cycles and less than 8 cycles.

### 6.8.3 How to Single-Step the 8051

One property of the 8051 Interrupt Structure enables single-stepping with very little software overhead. As previously noted, an interrupt request will not be responded to while an interrupt of equal priority level is still in progress, nor will it be responded to after RETI



## MCS-51 ARCHITECTURE

until at least one other instruction has been executed. Thus, once an interrupt routine has been entered, it cannot be re-entered until at least one instruction of the interrupted program is executed.

There are a number of ways that this feature can be used to single-step the 8051. One way is to program one of the external interrupts (say,  $\overline{\text{INT0}}$ ) to be level-activated. The service routine for this interrupt then will terminate with

```
JNB    P3.2,$      ;WAIT HERE TILL
                    INTO GOES HIGH
JB     P3.2,$      ;NOW WAIT HERE
                    TILL IT GOES LOW
RETI    ;GO BACK AND
                    EXECUTE ONE
                    INSTRUCTION.
```

Now if the  $\overline{\text{INT0}}$  pin, which is also the P3.2 pin, is held normally low, the CPU will go right into the External Interrupt 0 routine and stay there until  $\overline{\text{INT0}}$  is pulsed (from low to high to low). Then it will execute RETI, go back to the task program, execute one instruction, and immediately re-enter the External Interrupt 0 routine to await the next pulsing of P3.2. One step of the task program is executed each time P3.2 is pulsed.

### 6.8.4 Interrupt Control Registers

The Interrupt Request Flags are in two different registers and two port pins, as listed below:

| Source      | Request Flag                                   | Location |
|-------------|--|----------|
| External    | $\overline{\text{INT0}}$ , if $\text{IT0} = 0$ | P3.2     |
| Interrupt 0 | $\text{IE0}$ , if $\text{IT0} = 1$             | TCON.1   |
| Timer 0     | TF0  | TCON.5   |
| Overflow    |  |          |

|             |  |        |
|-------------|--|--------|
| External    | $\overline{\text{INT1}}$ , if $\text{IT1} = 0$ | P3.3   |
| Interrupt 1 | $\text{IE1}$ , if $\text{IT1} = 1$             | TCON.3 |
| Timer 1     | TF1  | TCON.7 |
| Overflow    |  |        |
| Serial Port | TI (on transmission)                           | SCON.1 |
|             | RI (on reception)                              | SCON.0 |

External Interrupt control bits IT0 and IT1 are in TCON.0 and TCON.2, respectively. Reset leaves all flags inactive, with IT0 and IT1 cleared.

All the interrupt flags can be set or cleared by software, with the same effect as by hardware.

The Enable and Priority Control Registers are shown below. All of these control bits are set or cleared by software. All are cleared by reset.

### IE: Interrupt Enable Register

|      |    |   |   |    |     |     |     |     |
|------|----|---|---|----|-----|-----|-----|-----|
| Bit: | 7  | 6 | 5 | 4  | 3   | 2   | 1   | 0   |
|      | EA | X | X | ES | ET1 | EX1 | ET0 | EX0 |

where

- EA disables all interrupts. If  $\text{EA} = 0$ , no interrupt will be acknowledged. If  $\text{EA} = 1$ , each interrupt source is individually enabled or disabled by setting or clearing its enable bit.
- ES enables or disables the Serial Port interrupt. If  $\text{ES} = 0$ , the Serial Port interrupt is disabled.
- ET1 enables or disables the Timer 1 Overflow interrupt. If  $\text{ET1} = 0$ , the Timer 1 interrupt is disabled.
- EX1 enables or disables External Interrupt 1. If  $\text{EX1} = 0$ , External Interrupt 1 is disabled.
- ET0 enables or disables the Timer 0 Overflow interrupt. If  $\text{ET0} = 0$ , the Timer 0 interrupt is disabled.



## MCS-51 ARCHITECTURE

### IP: Interrupt Priority Register

Bit: 7 6 5 4 3 2 1 0  
X X X PS PT1 PX1 PT0 PX0

where

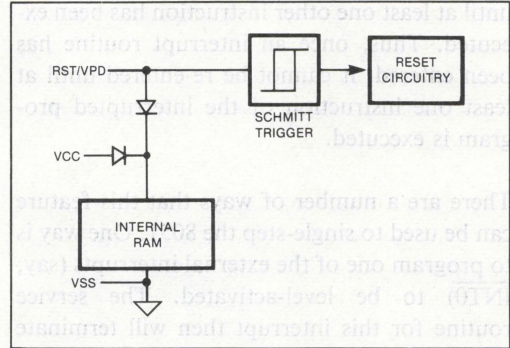
- PS defines the Serial Port interrupt priority level. PS = 1 programs it to the higher priority level.
- PT1 defines the Timer 1 interrupt priority level. PT1 = 1 programs it to the higher priority level.
- PX1 defines the External Interrupt 1 priority level. PX1 = 1 programs it to the higher priority level.
- PT0 defines the Timer 0 interrupt priority level. PT0 = 1 programs it to the higher priority level.
- PX0 defines the External Interrupt 0 priority level. PX0 = 1 programs it to the higher priority level.

## 6.9 RST/VPD PIN

The circuitry connected to the RST/VPD pin is shown in Figure 6-14. A Schmitt Trigger is used at the input to the reset circuitry for noise rejection. The output of the Schmitt Trigger is sampled by the reset circuitry at S5P2 of every machine cycle. At least two consecutive samples must show a high in order to effect a complete reset and initialization.

### 6.9.1 Reset

Reset is accomplished by holding the RST pin high for at least two machine cycles (24 oscillator periods) while the oscillator is running. The CPU responds by executing an inter-



**Figure 6-14. RST/VPD Circuitry**

nal reset. It also configures the ALE and  $\overline{\text{PSEN}}$  pins as inputs. (They are quasi-bidirectional.) The internal reset is executed during the second cycle in which RST is high and is repeated every cycle until RST goes low. It leaves the internal registers as follows:

| Register | Content       |
|----------|---------------|
| PC       | 0000H         |
| A        | 00H           |
| B        | 00H           |
| PSW      | 00H           |
| SP       | 07H           |
| DPTR     | 0000H         |
| P0 - P3  | 0FFH          |
| IP       | (XXX00000)    |
| IE       | (0XX00000)    |
| TMOD     | 00H           |
| TCON     | 00H           |
| TH0      | 00H           |
| TL0      | 00H           |
| TH1      | 00H           |
| TL1      | 00H           |
| SCON     | 00H           |
| SBUF     | Indeterminate |
| PCON     | (0XXXXXXX)    |



## MCS-51 ARCHITECTURE

The internal RAM is not affected by reset. When VCC is turned on, the RAM content is indeterminate unless VPD was applied prior to VCC being turned off (see Power Down Operation).

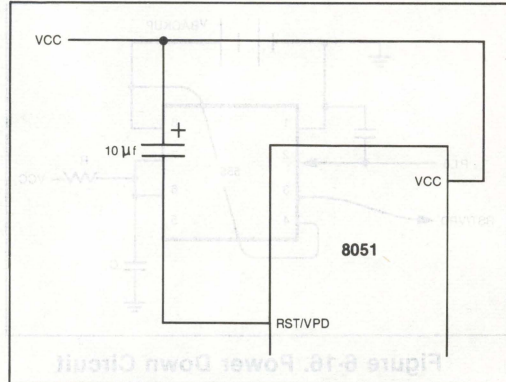
### POWER-ON RESET

An automatic reset can be obtained when VCC is turned on by connecting the RST pin to VCC through a  $10\ \mu\text{f}$  capacitor, providing the VCC risetime doesn't exceed a millisecond or so. A power-on reset circuit is shown in Figure 6-15. When power comes on the current drawn by RST commences to charge the capacitor. The voltage at RST is the difference between VCC and the capacitor voltage, and decreases from VCC as the cap charges. The larger the capacitor is, the more slowly VRST decreases. VRST must remain above the lower threshold of the Schmitt Trigger long enough to effect a complete reset. The time required is the oscillator start-up time plus 2 machine cycles. If the VCC risetime is less than 1 msec and the oscillator start-up time doesn't exceed 10 msec, a  $10\ \mu\text{f}$  capacitor will provide a reliable power-on reset.

### 6.9.2 Power Down Operation

During normal operation the internal RAM draws its power from VCC. However, as can be seen in Figure 6-14, if the voltage at RST/VPD exceeds VCC it becomes the source of power for the RAM. This allows a backup power supply to be used to hold RAM data in the event of a power failure.

To take advantage of this feature, the user's system, upon detecting that a power failure is imminent, would interrupt the processor via  $\overline{\text{INT0}}$  or  $\overline{\text{INT1}}$  to transfer relevant data to the RAM and enable the backup power supply to



**Figure 6-15. Power-On Reset**

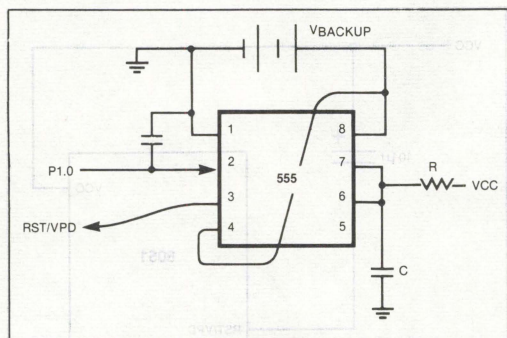
the RST/VPD pin before VCC falls below its operating limit. When power returns, VPD needs to stay on long enough to effect a reset (oscillator start-up time plus two machine cycles), and normal operation can resume.

Figure 6-16 suggests one possible implementation of the power-down feature. Assuming a detected imminent power failure interrupts the processor via  $\overline{\text{INT0}}$ , the External Interrupt 0 service routine transfers relevant data to the RAM and then writes a 0 to P1.0. The low at P1.0 triggers the 555, which is configured as a one-shot whose output pulse width depends on R, C, and the presence of VCC. If VCC is still present when the 555 times out, it is assumed that the "imminent power failure" was a false alarm, and operation resumes from reset. If VCC does in fact go down before the 555 times out, the 555 will hold power to RST/VPD during the outage, and will continue to hold it after VCC comes back, for a time determined by R and C. R and C should be selected so as to obtain a reliable power-on reset.

### 6.10 8751

The 8751 is the EPROM version of the 8051,





**Figure 6-16. Power Down Circuit**

that is, the on-chip Program Memory in the 8751 can be electrically programmed, and can be erased by exposure to ultraviolet light. Erasure leaves the array in an all 1s state.

## 6.10.1 Erasure Characteristics

Erasure of the 8751 Program Memory begins to occur when the chip is exposed to light with wavelengths shorter than approximately 4000 Angstroms. Since sunlight and fluorescent lighting have wavelengths in this range, constant exposure to these light sources over an extended period of time (about 1 week in sunlight, or 3 years in room-level fluorescent lighting) could cause unintentional erasure. If an application subjects the 8751 to this type of exposure, it is suggested that an opaque label be placed over the window. (Suitable labels are available from Intel.)

The recommended erasure procedure is exposure to ultraviolet light (at 2537 Angstroms) to an integrated dose of at least 15 W-sec/cm<sup>2</sup>. Exposing the 8751 to an ultraviolet lamp of 12000  $\mu$ W/cm<sup>2</sup> rating for 20 to 30 minutes, at a distance of about 1 inch, should be sufficient.

Erasure leaves the array in an all 1s state.

## 6.10.2 Programming the EPROM

To be programmed, the 8751 must be running with a 4 to 6 MHz oscillator. The address of an EPROM location to be programmed is applied to Port 1 and pins P2.0-P2.3 of Port 2, while the data byte is applied to Port 0. Pins P2.4-P2.6 and PSEN should be held low, and P2.7 and RST high. (These are TTL levels, except RST which requires 2.5V for a high.)  $\overline{EA}$  is held normally at TTL high, and is pulsed to +21V. While  $\overline{EA}$  is at 21V, the ALE pin, which is normally being held at TTL high, is pulsed low for 50 msec. Then  $\overline{EA}$  is returned to TTL high. This is illustrated in Figure 6-17. Detailed timing specifications are given in the data sheets.

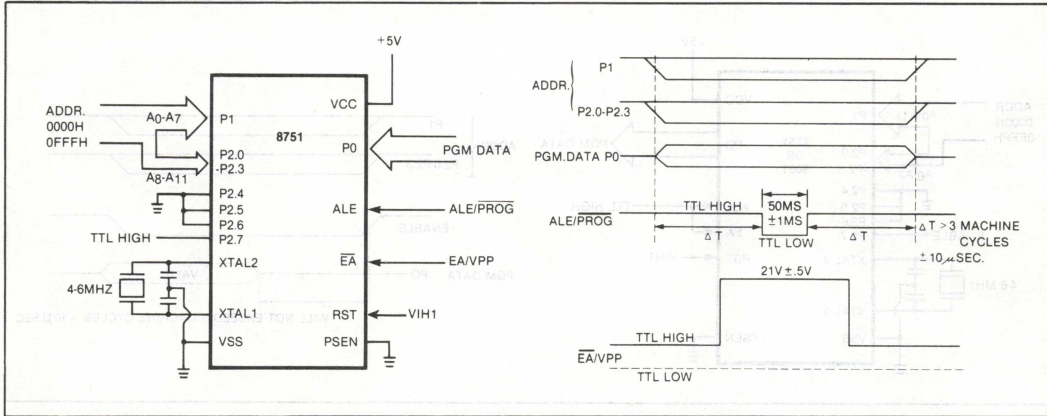
Note: The  $\overline{EA}$  pin must not be allowed to go above the maximum specified VPP level of 21.5 V, even instantaneously. Even a narrow glitch above that voltage level can cause permanent damage. It is suggested that the VPP source be well regulated and free of glitches.

## 6.10.3 Program Verification

Refer to Figure 6-18 for the Program Verification setup for the 8751 or the 8051. To read the Program Memory of either the 8751 or the 8051, the following procedure can be used. The unit must be running with a 4 to 6 MHz oscillator. The address of a Program Memory location to be read is applied to Port 1 and pins P2.0-P2.3 of Port 2. Pins P2.4-P2.6 and PSEN are held low, while the ALE, RST, and  $\overline{EA}$  pins are held high. (These are TTL levels except RST, which requires 2.5V for a high.) Port 0 will be the data output lines. P2.7 can be used as a read strobe. While P2.7 is held at TTL high, the Port 0 pins float. When P2.7 is strobed low, the contents of the addressed location will appear at Port 0. External pull-



## MCS-51 ARCHITECTURE



**Figure 6-17. Programming the 8751**

ups (e.g., 10K) are required on Port 0 during this operation.

### 6.11 8051 FAMILY PIN DESCRIPTION

**VSS:** Circuit ground potential.

**VCC:** Supply voltage during programming (of the 8751), verification (of the 8051 or 8751), and normal operation.

**Port 0:** Port 0 is an 8-bit open drain bidirectional I/O port. It is also the multiplexed low-order address and data bus during accesses to external memory (during which accesses it activates internal pullups). It also outputs instruction bytes during program verification. (External pullups are required during program verification.) Port 0 can sink eight LS TTL inputs.

**Port 1:** Port 1 is an 8-bit bidirectional I/O port with internal pullups. It receives the low-order address byte during program verification in the 8051 or 8751. Port 1 can sink/source three LS TTL inputs. It can drive MOS inputs without external pullups.

**Port 2:** Port 2 is an 8-bit bidirectional I/O port

with internal pullups. It emits the high-order address byte during accesses to external memory. It also receives the high-order address bits and control signals during program verification in the 8051 or 8751. Port 2 can sink/source three LS TTL inputs. It can drive MOS inputs without external pullups.

**Port 3:** Port 3 is an 8-bit bidirectional I/O port with internal pullups. It also serves the functions of various special features of the MCS-51 Family, as listed below:

| Port Pin | Alternate Function   |
|----------|--|
| P3.0     | RXD (serial input port)                                    |
| P3.1     | TXD (serial output port)                                   |
| P3.2     | $\overline{\text{INT0}}$ (external interrupt)              |
| P3.3     | $\overline{\text{INT1}}$ (external interrupt)              |
| P3.4     | T0 (Timer 0 external input)                                |
| P3.5     | T1 (Timer 1 external input)                                |
| P3.6     | $\overline{\text{WR}}$ (external Data Memory write strobe) |
| P3.7     | $\overline{\text{RD}}$ (external Data Memory read strobe)  |



## MCS-51 ARCHITECTURE

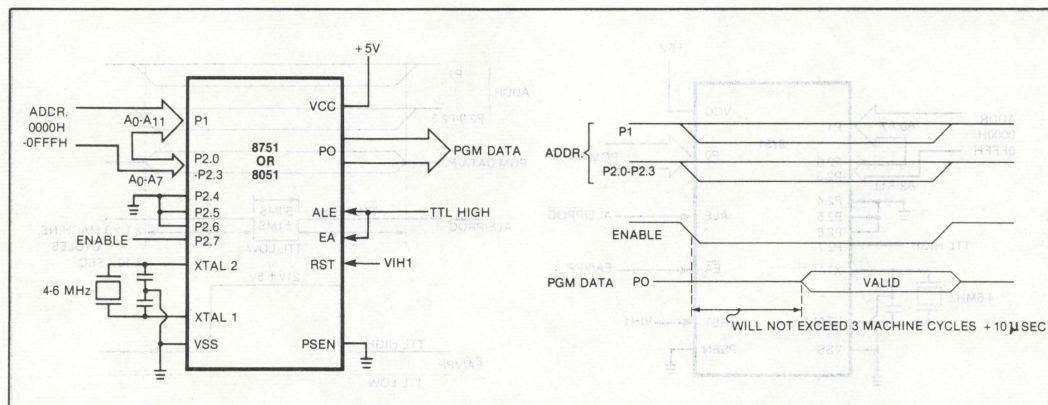


Figure 6-18. Program Verification

Port 3 can sink/source three LS TTL inputs. It can drive MOS inputs without external pullups.

**RST/VPD:** A high level on this pin for two machine cycles while the oscillator is running resets the device. An internal pulldown permits Power-On reset using only a capacitor connected to VCC.

**ALE/PROG:** Address Latch Enable output for latching the low byte of the address during accesses to external memory. ALE is activated though for this purpose at a constant rate of 1/6 the oscillator frequency even when external memory is not being accessed. Consequently it can be used for external clocking or timing purposes. (However, one ALE pulse is skipped during each access to external Data Memory.) This pin is also the program pulse input (PROG) during EPROM programming.

**PSEN:** Program Store Enable output is the read strobe to external Program Memory. PSEN is activated twice each machine cycle during fetches from external Program Memory. (However, when executing out of external Program Memory two activations of PSEN are skipped during each access to external

Program Memory.) PSEN is not activated during fetches from internal Program Memory.

**$\overline{EA}/VPP$ :** When EA is held high the CPU executes out of internal Program Memory (unless the Program Counter exceeds 0FFFH). When  $\overline{EA}$  is held low the CPU executes only out of external Program Memory. In the 8031,  $\overline{EA}$  must be externally wired low. In the 8751, this pin also receives the 21V programming supply voltage (VPP) during EPROM programming.

**XTAL1:** Input to the inverting amplifier that forms the oscillator. Should be grounded when an external oscillator is used.

**XTAL2:** Output of the inverting amplifier that forms the oscillator, and input to the internal clock generator. Receives the external oscillator signal when an external oscillator is used.



---

# **MCS<sup>®</sup>-51 Memory Organization, Addressing Modes, and Data Manipulation**

---

**7**



## CHAPTER 7

# MCS<sup>®</sup>-51 MEMORY, ORGANIZATION, ADDRESSING MODES AND DATA MANIPULATION

### 7.0 MEMORY ORGANIZATION

In the 8051 family the memory is organized over three address spaces and the program counter. The memory spaces shown in Figure 6-2 (page 6-3) are the:

- 64K-byte Program Memory address space
- 64K-byte External Data Memory address space
- 384-byte Internal Data Memory address space

The 16-bit Program Counter register provides the 8051 with its 64K addressing capabilities. The Program Counter allows the user to execute calls and branches to any location within the Program Memory space. There are no instructions that permit program execution to move from the Program Memory space to any of the data memory spaces.

In the 8051 and 8751 the lower 4K of the 64K Program Memory address space is filled by internal ROM and EPROM, respectively. By tying the  $\overline{EA}$  pin high, the processor can be forced to fetch from the internal ROM/EPROM for Program Memory addresses 0 through 4K. Bus expansion for accessing Program Memory beyond 4K is automatic since external instruction fetches occur automatically when the Program Counter increases above 4095. If the  $\overline{EA}$  pin is tied low all Program Memory fetches are from external memory. The execution speed of the 8051 is the same regardless of whether fetches are from internal or external Program Memory. If

all program storage is on-chip, byte location 4095 should be left vacant to prevent an undesired prefetch from external Program Memory address 4096.

Certain locations in Program Memory are reserved for specific programs. Locations 0000 through 0002 are reserved for the initialization program. Following reset, the CPU always begins execution at location 0000. Locations 0003 through 0042 are reserved for the five interrupt-request service programs. Each resource that can request an interrupt requires that its service program be stored at its reserved location.

The 64K-byte External Data Memory address space is automatically accessed when the MOVX instruction is executed.

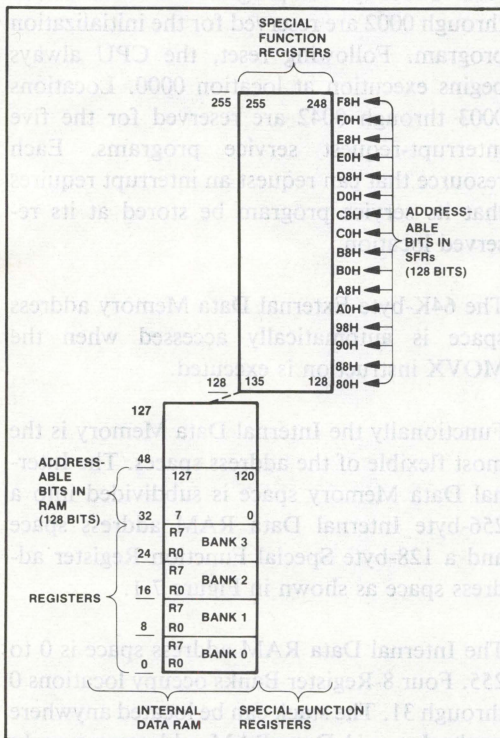
Functionally the Internal Data Memory is the most flexible of the address spaces. The Internal Data Memory space is subdivided into a 256-byte Internal Data RAM address space and a 128-byte Special Function Register address space as shown in Figure 7-1.

The Internal Data RAM address space is 0 to 255. Four 8-Register Banks occupy locations 0 through 31. The stack can be located anywhere in the Internal Data RAM address space. In addition, 128 bit locations of the on-chip RAM are accessible through Direct Addressing. These bits reside in Internal Data RAM at byte locations 32 through 47. Currently locations 0 through 127 of the Internal Data RAM address space are filled with on-chip RAM.



## MCS-51 MEMORY, ADDRESSING, DATA MANIPULATION

The stack depth is limited only by the available Internal Data RAM, thanks to an 8-bit reloadable Stack Pointer. The stack is used for storing the Program Counter during subroutine calls and may be used for passing parameters. Any byte of Internal Data RAM or Special Function Register accessible through Direct Addressing can be pushed/popped.



**Figure 7-1. Internal Data Memory Address Space**

The Special Function Register address space is 128 to 255. All registers except the Program Counter and the four 8-Register Banks reside here. Memory mapping the Special Function Registers allows them to be accessed as easily as internal RAM. As such, they can be operated on by most instructions. In addition, 128 bit locations within the Special Function Register address space can be accessed using Direct Addressing. These bits reside in the Special Function Register byte locations divisible by eight. The twenty Special Function Registers are listed in Figure 7-2. Their mapping in the Special Function Register address space is shown in Figures 7-3 and 7-4.

Performing a read from a location of the Internal Data memory where neither a byte of Internal Data RAM (i.e., RAM addresses 128-255) nor a Special Function Register exists will access data of indeterminable value.

Architecturally, each memory space is a linear sequence of 8-bit wide bytes. By Intel convention the storage of multi-byte address and data operands in program and data memories is the least significant byte at the low-order address and the most significant byte at the high-order address. Within byte X, the most significant bit is represented by X.7 while the least significant bit is X.0. Any deviation from these conventions will be explicitly stated in the text.



## MCS-51 MEMORY, ADDRESSING, DATA MANIPULATION

### ARITHMETIC REGISTERS:

ACCumulator\*, B register\*,  
Program Status Word\*

### POINTERS:

Stack Pointer, Data Pointer (high & low)

### PARALLEL I/O PORTS:

Port 3\*, Port 2\*, Port 1\*, Port 0\*

### INTERRUPT SYSTEM:

Interrupt Priority Control\*,  
Interrupt Enable Control\*

### TIMERS:

Timer MODE, Timer CONTROL\*, Timer 1  
(high & low), Timer 0 (high & low)

### SERIAL I/O PORT:

Serial CONTROL\*, Serial data BUFFER,  
PCON

\*Bits in these registers are bit addressable

Figure 7-2. Special Function Registers

## 7.1 OPERAND ADDRESSING

There are five methods of addressing source operands. They are Register Addressing, Direct Addressing, Register-Indirect Addressing, Immediate Addressing and Base-Register-plus Index-Register-Indirect Addressing. The first three of these methods can also be used to address a destination operand. Since operations in the 8051 require 0 (NOP only), 1, 2, 3 or 4 operands, these five addressing methods are used in combinations to provide the 8051 with its 21 addressing modes.

Most instructions have a "destination, source" field that specifies the data type, addressing methods and operands involved. For operations other than moves, the destination operand is also a source operand. For example, in "subtract-with-borrow A,#5" the A register receives the result of the value in register A minus 5, minus C.

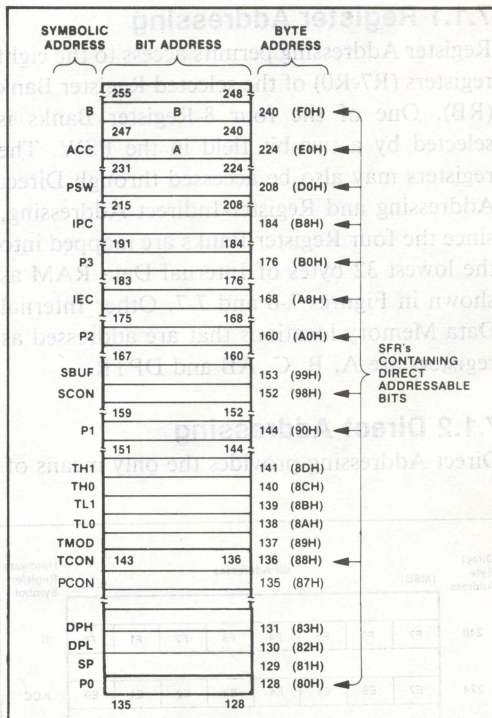


Figure 7-3. Mapping of Special Function Registers

Most operations involve operands that are located in Internal Data Memory. The selection of the Program Memory space or External Data Memory space for a second operand is determined by the operation mnemonic unless it is an immediate operand. The subset of the Internal Data Memory being addressed is determined by the addressing method and address value. For example, the Special Function Registers can be accessed only through Direct Addressing with an address of 128-255. A summary of the operand addressing methods is shown in Figure 7-5. The following paragraphs describe the five addressing methods.



## MCS-51 MEMORY, ADDRESSING, DATA MANIPULATION

### 7.1.1 Register Addressing

Register Addressing permits access to the eight registers (R7-R0) of the selected Register Bank (RB). One of the four 8-Register Banks is selected by a two-bit field in the PSW. The registers may also be accessed through Direct Addressing and Register-Indirect Addressing, since the four Register Banks are mapped into the lowest 32 bytes of Internal Data RAM as shown in Figures 7-6 and 7-7. Other Internal Data Memory locations that are addressed as registers are A, B, C, AB and DPTR.

### 7.1.2 Direct Addressing

Direct Addressing provides the only means of

| Direct<br>Byte<br>Address | Bit Addresses |     |     |     |     |     |     |     | Hardware<br>Register<br>Symbol |  |
|---------------------------|---------------|-----|-----|-----|-----|-----|-----|-----|--------------------------------|--|
|                           | (MSB)         |     |     |     |     |     |     |     | (LSB)                          |  |
| 240                       | F7            | F6  | F5  | F4  | F3  | F2  | F1  | F0  | B                              |  |
| 224                       | E7            | E6  | E5  | E4  | E3  | E2  | E1  | E0  | ACC                            |  |
| 208                       | CY            | AC  | FO  | RS1 | RS0 | OV  | P   |     | PSW                            |  |
|                           | D7            | D6  | D5  | D4  | D3  | D2  | D1  | D0  |                                |  |
| 184                       |               |     |     | PS  | PT1 | PX1 | PT0 | PX0 | IP                             |  |
|                           | —             | —   | —   | BC  | BB  | BA  | B9  | B8  |                                |  |
| 176                       | B7            | B6  | B5  | B4  | B3  | B2  | B1  | B0  | P3                             |  |
| 168                       | EA            |     |     | ES  | ET1 | EX1 | ET0 | EX0 | IE                             |  |
|                           | AF            | —   | —   | AC  | AB  | AA  | A9  | A8  |                                |  |
| 160                       | A7            | A6  | A5  | A4  | A3  | A2  | A1  | A0  | P2                             |  |
| 152                       | SM0           | SM1 | SM2 | REN | TB8 | RB8 | T1  | RI  | SCON                           |  |
|                           | 9F            | 9E  | 9D  | 9C  | 9B  | 9A  | 99  | 98  |                                |  |
| 144                       | 97            | 96  | 95  | 94  | 93  | 92  | 91  | 90  | P1                             |  |
| 136                       | TF1           | TR1 | TF0 | TR0 | IE1 | IT1 | IE0 | IT0 | TCON                           |  |
|                           | 8F            | 8E  | 8D  | 8C  | 8B  | 8A  | 89  | 88  |                                |  |
| 128                       | 87            | 86  | 85  | 84  | 83  | 82  | 81  | 80  | P0                             |  |

Figure 7-4. Special Function Register Bit Address

accessing the memory-mapped byte-wide Special Function Registers and memory mapped bits within the Special Function Registers and Internal Data RAM. Direct Addressing of bytes may also be used to access the lower 128 bytes of Internal Data RAM. Direct Addressing of bits gains access to a 128 bit subset of the Internal Data RAM and 128 bit subset of the Special Function Registers as shown in Figures 7-3, 7-4, 7-6, and 7-7.

Register-Indirect Addressing using the content of R1 or R0 in the selected Register Bank, or

- **Register Addressing**
  - R7-R0
  - A,B,C (bit), AB (two bytes), DPTR (double byte)
- **Direct Addressing**
  - Lower 128 bytes of Internal Data RAM
  - Special Function Registers
  - 128 bits in subset of Internal Data RAM address space
  - 128 bits in subset of Special Function Register address space
- **Register-Indirect Addressing**
  - Internal Data RAM [ $@R1$ ,  $@R0$ ,  $@SP$  (PUSH and POP only)]
  - Least Significant Nibbles in Internal Data RAM ( $@R1$ ,  $@R0$ )
  - External Data Memory ( $@R1$ ,  $@R0$ ,  $@DPTR$ )
- **Immediate Addressing**
  - Program Memory (in-code constant)
- **Base-Register plus Index-Register Indirect Addressing**
  - Program Memory ( $@DPTR + A$ ,  $@PC + A$ )

Figure 7-5. Operand Addressing Methods



## MCS-51 MEMORY, ADDRESSING, DATA MANIPULATION

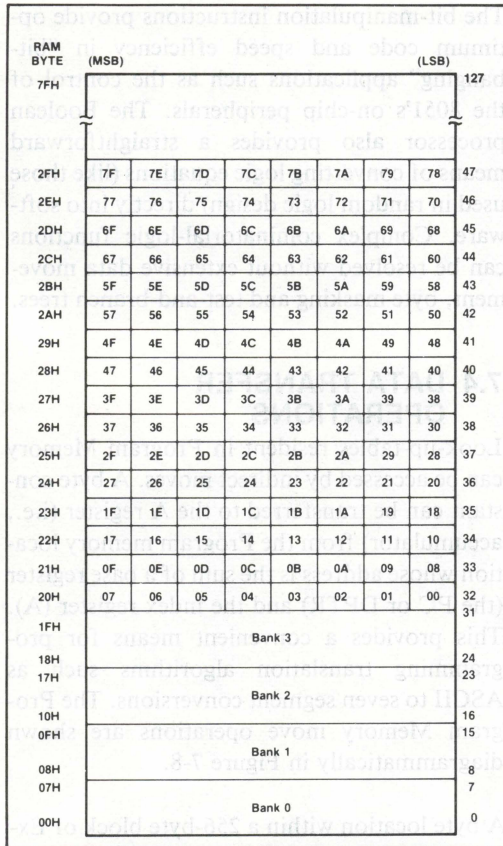


Figure 7-6. RAM Bit Addresses

using the content of the Stack Pointer (PUSH and POP only), addresses the Internal Data RAM. Register-Indirect Addressing is also used for accessing the External Data Memory. In this case, either R1 or R0 in the selected Register Bank may be used for accessing locations within a 256-byte block. The block number can be preselected by the contents of a port. The 16-bit Data Pointer may be used for accessing any location within the full 64K external address space.

### 7.1.3 Immediate Addressing

Immediate Addressing allows constants which

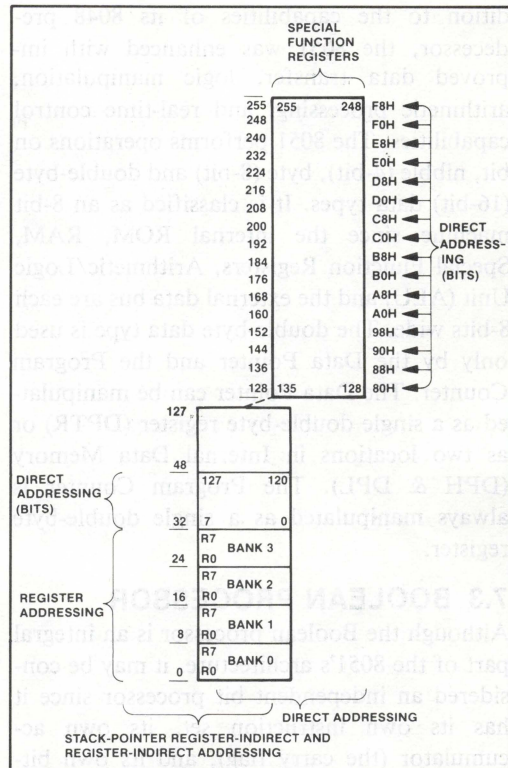


Figure 7-7. Addressing Operands in Internal Data Memory

are part of the instruction to be accessed from the Program Memory.

### 7.1.4 Base-Register- plus Index-Register- Indirect Addressing

Base-Register- plus Index-Register- Indirect Addressing simplifies accessing look-up-tables (LUT) resident in Program memory. A byte may be accessed from a LUT via an indirect move from a location whose address is the sum of a base register (the DPTR or PC) and the index register (A).

## 7.2 DATA MANIPULATION

The 8051 microcomputer is efficient both as an arithmetic processor and as a controller. In ad-



dition to the capabilities of its 8048 predecessor, the 8051 was enhanced with improved data transfer, logic manipulation, arithmetic processing, and real-time control capabilities. The 8051 performs operations on bit, nibble (4-bit), byte (8-bit) and double-byte (16-bit) data types. It is classified as an 8-bit machine since the internal ROM, RAM, Special Function Registers, Arithmetic/Logic Unit (ALU) and the external data bus are each 8-bits wide. The double-byte data type is used only by the Data Pointer and the Program Counter. The Data Pointer can be manipulated as a single double-byte register (DPTR) or as two locations in Internal Data Memory (DPH & DPL). The Program Counter is always manipulated as a single double-byte register.

### 7.3 BOOLEAN PROCESSOR

Although the Boolean processor is an integral part of the 8051's architecture, it may be considered an independent bit processor since it has its own instruction set, its own accumulator (the carry flag), and its own bit-addressable RAM and I/O.

The bit-manipulation instructions allow the Direct Addressing of 128 bits within the Internal Data RAM and 128 bits within the Special Function Registers. The Special Function Registers with an address evenly divisible by eight (P0, TCON, P1, SCON, P2, IEC, P3, IPC, PSW, A, and B) contain Direct Addressable bits. On any addressable bit, the Boolean processor can perform the bit operations of set, clear, complement, jump-if-set, jump-if-not-set, jump-if-set-then-clear and move to/from carry. Between any addressable bit (or its complement) and the carry flag it can perform the bit operation of logical and or logical or with the result returned to the carry flag.

The bit-manipulation instructions provide optimum code and speed efficiency in "bit-banging" applications such as the control of the 8051's on-chip peripherals. The Boolean processor also provides a straightforward means of converting logic equations (like those used in random logic design) directly into software. Complex combinatorial-logic functions can be resolved without extensive data movement, byte masking and test-and-branch trees.

### 7.4 DATA TRANSFER OPERATIONS

Look-up-tables resident in Program Memory can be accessed by indirect moves. A byte constant can be transferred to the A register (i.e., accumulator) from the Program memory location whose address is the sum of a base register (the PC or DPTR) and the index register (A). This provides a convenient means for programming translation algorithms such as ASCII to seven segment conversions. The Program Memory move operations are shown diagrammatically in Figure 7-8.

A byte location within a 256-byte block of External Data Memory can be accessed using R1 or R0 in Register-Indirect Addressing. Any location within the full 64K External Data Memory address space can be accessed through Register-Indirect Addressing using a 16-bit base register (i.e., the Data Pointer). These moves are shown in Figure 7-9.

The byte in-code-constant (immediate) moves and byte variable moves within the 8051 are highly orthogonal as detailed in Figure 7-10. When one considers that the accumulator and the registers in the Register Banks can be Direct Addressed, the two-operand data transfer operations allow a byte to be moved between any two of the RB registers, Internal



## MCS-51 MEMORY, ADDRESSING, DATA MANIPULATION

Data RAM, accumulator and Special Function Registers. Also, immediate operands can be moved to any of these locations. Of particular interest is the Direct Address to Direct Address move which permits the value in a port to be moved to the Internal Data RAM without using any RB registers or the accumulator. The Data Pointer register can be loaded with a double-byte immediate value. Also, the 8051's Boolean Processor can move any Direct Ad-

dressed bit to or from the carry register.

The A register can be exchanged with a register in the selected Register Bank, with a Register-Indirect Addressed byte in the Internal Data RAM or with a Direct Addressed byte in the Internal Data RAM or Special Function Register. The least significant nibble of the A register can also be exchanged with the least

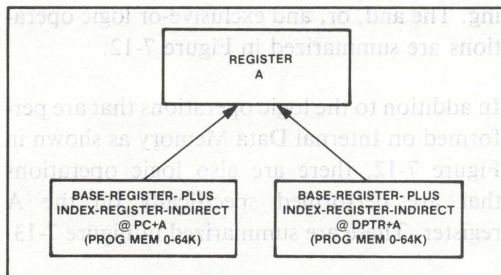


Figure 7-8. Program Memory Move Operations

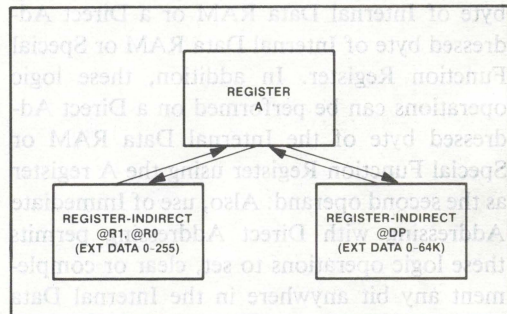


Figure 7-9. External Data Memory Move Operations

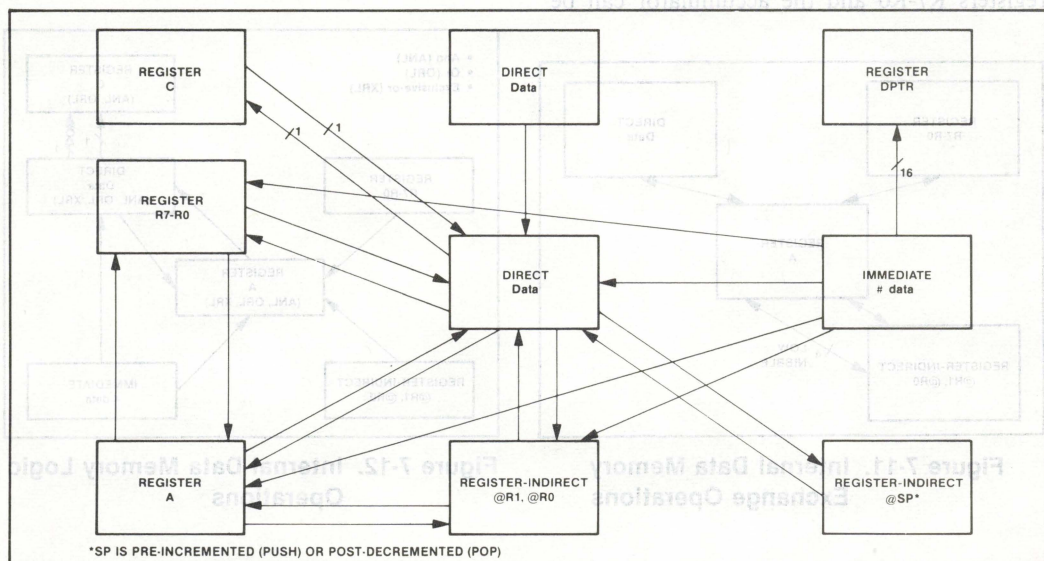


Figure 7-10. Internal Data Memory Move Operations



## MCS-51 MEMORY, ADDRESSING, DATA MANIPULATION

significant nibble of a Register-Indirect Addressed byte in the Internal Data RAM. The exchange operation is shown in Figure 7-11.

### 7.5 LOGIC OPERATIONS

The 8051 permits the logic operations of and, or, and exclusive-or to be performed on the A register by a second operand which can be immediate value, a register in the selected Register Bank, a Register-Indirect Addressed byte of Internal Data RAM or a Direct Addressed byte of Internal Data RAM or Special Function Register. In addition, these logic operations can be performed on a Direct Addressed byte of the Internal Data RAM or Special Function Register using the A register as the second operand. Also, use of Immediate Addressing with Direct Addressing permits these logic operations to set, clear or complement any bit anywhere in the Internal Data RAM or Special Function Registers without affecting the PSW, RB registers or accumulator. When one takes into account that registers R7-R0 and the accumulator can be

Direct Addressed, the two-operand logic operations allow the destination (first operand) to be a byte in the Internal Data RAM, a Special Function Register, RB registers (R7-R0) or the accumulator while the choice of the second operand can be any of the aforementioned or an immediate value. The 8051 can also perform a logical or, or a logical and, between the Boolean accumulator (i.e., the carry flag) and any bit, or its complement, that can be accessed through Direct Addressing. The and, or, and exclusive-or logic operations are summarized in Figure 7-12.

In addition to the logic operations that are performed on Internal Data Memory as shown in Figure 7-12, there are also logic operations that are performed specifically on the A register. These are summarized in Figure 7-13.

In addition to the and and or bit logical shown in Figure 7-12, there are logicals that can operate exclusively on a Direct Addressed bit.

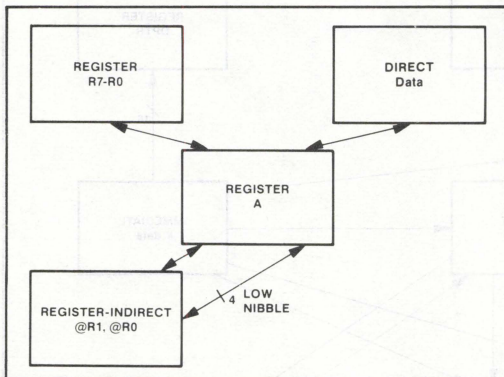


Figure 7-11. Internal Data Memory Exchange Operations

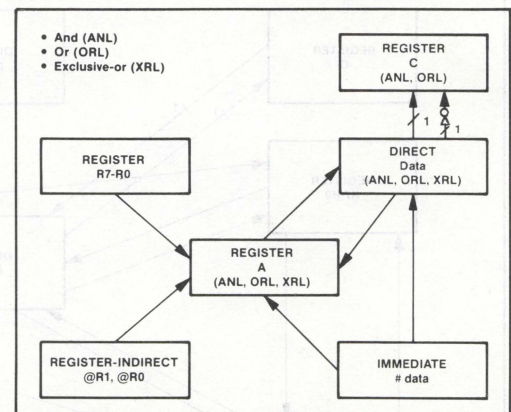


Figure 7-12. Internal Data Memory Logic Operations



## MCS-51 MEMORY, ADDRESSING, DATA MANIPULATION

These operations are listed in Figure 7-14. The carry flag is also addressed as a register and can be set, cleared or complemented with one-byte instructions.

### 7.6 ARITHMETIC OPERATIONS

Along with the existing 8048 arithmetic operations of add, increment, decrement, compare-to-zero, decrement-and-compare-to-zero, and decimal-add-adjust, the 8051 implemented subtract-with-borrow, compare, multiply and divide.

Only unsigned binary integer arithmetic is performed in the Arithmetic/Logic Unit. In the two-operand operations of add, add-with-carry and subtract-with-borrow, the A register is the first operand and receives the result of the operation. The second operand can be an immediate byte, a register in the selected Register Bank, a Register-Indirect Addressed byte or a Direct Addressed byte. These instructions affect the overflow, carry, auxiliary-carry and parity flag in the Program Status Word (PSW). The carry flag facilitates non-signed integer and multi-precision addition

and subtraction and multi-precision rotation. Handling two's-complement-integer (signed) addition and subtraction can easily be accommodated with software's monitoring of the PSW's overflow flag. The auxiliary-carry flag simplifies BCD arithmetic. An operation that has an arithmetic aspect similar to a subtract is the compare-and-jump-if-not-equal operation. This operation performs a conditional branch if a register in the selected Register Bank, or an Indirect Addressed byte of Internal Data RAM, does not equal an immediate value; or if the A register does not equal a byte in the Direct Addressable Internal Data RAM, or a Special Function Register. While the destination operand is not updated and neither source operand is affected by the compare operation, the carry flag is. A summary of the two-operand add/subtract operations is shown in Figure 7-15.

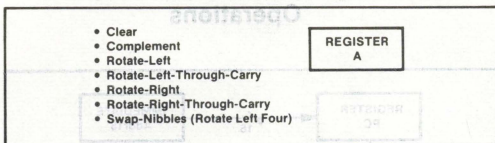


Figure 7-13. Internal Data Memory Logic Operations (Register A Specific)

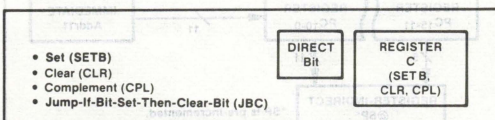


Figure 7-14. Internal Data Memory Logic Operations (Bit-Specific)

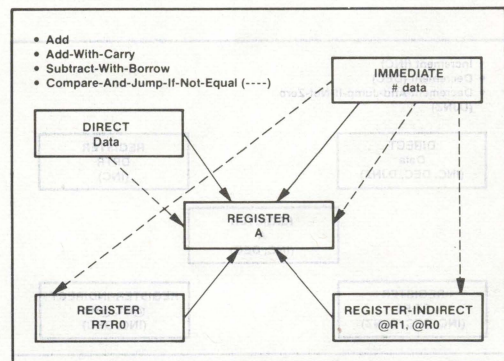


Figure 7-15. Internal Data Memory Arithmetic Operations

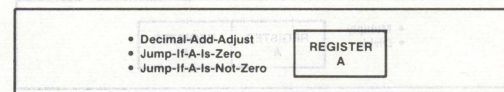


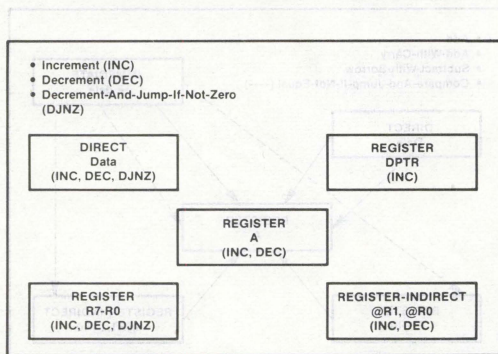
Figure 7-16. Internal Data Memory Arithmetic Operations (Register A Specific)



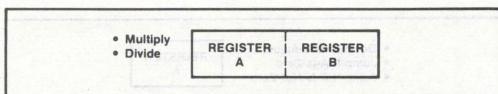
## MCS-51 MEMORY, ADDRESSING, DATA MANIPULATION

There are three arithmetic operations that operate exclusively on the A register. These are the decimal-adjust for BCD addition and the two test conditions shown in Figure 7-16. The decimal-adjust operation converts the result from a binary addition to two two-digit BCD values to yield the correct two-digit BCD result. During this operation the auxiliary-carry flag helps effect the proper adjustment. Conditional branches may be taken based on the value in the A register being zero or not zero.

The 8051 simplifies the implementation of software counters since the increment and decrement operations can be performed on the A register, a register in the selected Register Bank, an Indirect Addressed byte in the Internal Data RAM or a byte in the Direct Addressed Internal Data RAM or Special Function Register. The 16-bit Data Pointer can be



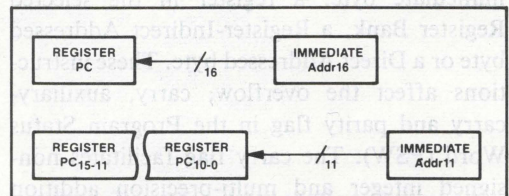
**Figure 7-17. Internal Data Memory Arithmetic Operations**



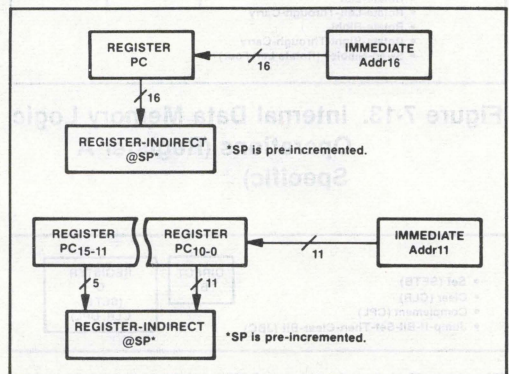
**Figure 7-18. Internal Data Memory Arithmetic Operations (Register A with B Specific)**

incremented. For efficient loop control the decrement-and-jump-if-not-zero operation is provided. This operation can test a register in the selected Register Bank, any Special Function Register or any byte of Internal Data RAM accessible through Direct Addressing and force a branch if it is not zero. The increment/decrement operations are summarized in Figure 7-17.

The multiply operation multiplies the one-byte A register by the one-byte B register and returns a double-byte result (MSB in B, LSB in A). The divide operation divides the one-byte A register by the one-byte B register and returns a byte quotient to the A register and a byte remainder to the B register. These are shown in Figure 7-18.



**Figure 7-19. Unconditional Branch Operations**



**Figure 7-20. Call Operations**



## MCS-51 MEMORY, ADDRESSING, DATA MANIPULATION

### 7.7 CONTROL TRANSFER

The 8051 has a non-paged Program Memory to accommodate relocatable code. The advantage of a non-paged memory is that a minor change to a program that causes a shift of the code's position in memory will not cause page boundary readjustments to be necessary. This also makes relocation possible. Relocation is desirable since it permits several programmers to write relocatable modules in various assembly and high-level languages which can later be linked together to form the machine object code.

Sixteen-bit jumps and calls are provided to allow branching to any location in the contiguous 64K Program Memory address space and preempt the need for Program Memory bank switching. Eleven-bit jumps and calls are also provided to maintain compatibility with the 8048 and to provide an efficient jump within a 2K program module. Unlike the 8048,

the 8051's call operations do not push the Program Status Word (PSW) to the stack along with the Program Counter, since many subroutines written for the 8051 do not affect the PSW. Hence the 8051 return operations pop only the Program Counter. The 8051's branch, call and return operations are shown diagrammatically in Figures 7-19, 7-20, and 7-21, respectively.

The 8051 also provides a method for performing conditional and unconditional branching relative to the starting address of the next instruction (PC - 128 to PC + 127). The bit test operations allow a conditional branch to be taken on the condition of a Direct Addressed bit being set or not set. The accumulator test operations allow a conditional branch based on the accumulator being zero or non-zero. Also provided are compare-and-jump-if-not-equal and decrement-and-compare-to-zero. These are shown in Figure 7-22.

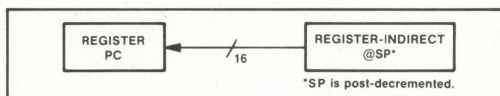


Figure 7-21. Return Operation

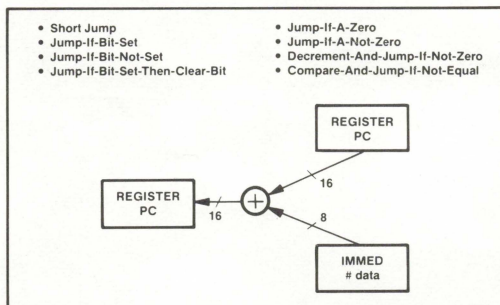


Figure 7-22. Unconditional Short Branch and Conditional Branch Operations

The register-indirect jump in the 8051 permits branching relative to a base register (DPTR) with an offset provided by the non-signed integer value in the index register (A). This accommodates N-way branching. The indirect jump is shown in Figure 7-23.

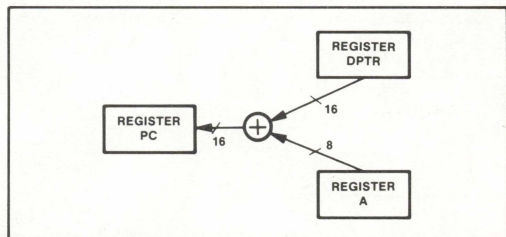


Figure 7-23. Unconditional Branch (Indirect) Operation







---

# MCS<sup>®</sup>-51 Instruction Set

8

---



## CHAPTER 8

# MCS®-51 INSTRUCTION SET

### 8.0 WHAT THE INSTRUCTION SET IS

An instruction set is a set of codes that directs a computer to perform its operations. The ease of understanding the instruction set does not depend upon the structure of the machine codes that the computer recognizes, so much as it depends upon the structure of the symbolic language that is used to describe the machine codes.

The 8051 assembly language needs only forty-two mnemonics to specify the 8051's thirty-three functions. A function may have several mnemonics (e.g., MOV, MOVX, MOVC) since the function mnemonic specifies when the Program Memory or External Data Memory is used in conjunction with the Internal Data Memory. When the function mnemonics are combined with unique address combinations specified in the "destination, source" field, 111 instructions are possible. The "destination, source" field specifies the data type and the combination of addressing methods to be used to address the destination and source operands. A summary of the 8051 instruction set is provided in Table 8-1.

The syntax of most 8051 assembly language instructions consists of a function mnemonic followed by a "destination, source" operand field. Thus "MOV @R0, Data" may be interpreted as "The content of the Internal Data Memory location addressed by the content of Register 0 receives the content of the Internal Data Memory location addressed by Data." In two operand instructions, the destination address also serves as the address of the first source. As an example of this, "ANL Data,

#5" may be interpreted as "The content of the Internal Data Memory location addressed by Data receives the result of the operation when the content of the memory location specified by Data is and-ed with the immediate 5."

The 8051's instruction set is an enhancement of the instruction set familiar to MCS-48 users. It is enhanced to allow expansion of on-chip CPU peripherals and to optimize byte efficiency and execution speed. Efficient use of program memory results from an instruction set consisting of 49 single-byte, 45 two-byte and 17 three-byte instructions. Most arithmetic, logical and branching operations can be performed using an instruction that appends either a short address or a long address. For example, Register Addressing allows a two-byte equivalent of the three byte Direct Addressing instructions. Also, short branches are more code efficient than long branches. 64 instructions execute in twelve oscillator periods, 45 instructions execute in twenty-four oscillator periods, and multiply and divide take only forty-eight oscillator periods. The number of bytes in each instruction and the number of oscillator periods required for execution are listed in Table 8-1.

### 8.1 ORGANIZATION OF THE INSTRUCTION SET

Instructions are described here in four functional groups:

- Data Transfer
- Arithmetic
- Logic
- Control Transfer



## MCS-51 INSTRUCTION SET

---

The Data Transfer, Arithmetic and Logic groups mentioned in the preceding list are further subdivided into an array of codes that specify whether the operation is to act upon immediate, RB register, accumulator, SFR or memory locations; whether bits, nibbles, bytes or double-bytes are to be processed; and what addressing methods are to be employed.

### 8.1.1 Data Transfer

Data transfer operations are divided into three classes:

- General Purpose
- Accumulator-Specific
- Address-Object

None affect the flag settings except a POP or MOV into the PSW.

#### General-Purpose Transfers

Three general-purpose data transfer operations are provided. These may be applied to most operands, though there are specific exceptions.

- MOV performs a bit or a byte transfer from the source operand to the destination operand.
- PUSH increments the SP register and then transfers a byte from the source operand to the stack element currently addressed by SP.
- POP transfers a byte operand from the stack element addressed by the SP register to the destination operand and then decrements SP.

#### Accumulator-Specific Transfers

Four accumulator-specific transfer operations are provided:

- XCH exchanges the byte source operand with register A (accumulator).
- XCHD exchanges the low-order nibble of the byte source operand with the low-order nibble of register A.
- MOVX performs a byte move between the External Data Memory and the A register. The external address can be specified by the DPTR register (16-bit) or the R1 or R0 register (8-bit).
- MOVC performs the move of a byte from the Program Memory to register A as follows. The operand in the A register is used as an index into a 256-byte table pointed to by the base register (DPTR or PC). The byte operand accessed is transferred to A. MOVC is used for table-look-up byte translation and for accessing operands from code-in-line tables.

#### Address-Object Transfer

- MOV DPTR,#data loads 16-bits of immediate data into a pair of destination registers, DPH and DPL (DHP from low-order address, DPL from high-order address).

### 8.1.2 Logic

The 8051 performs the basic logic operations on both bit and byte operands.

#### Single-Operand Operations

Seven single-operand logical operations are provided:

- CLR is used to set either the A register, the C register, or any Direct Addressed bit to zero (0).



## MCS-51 INSTRUCTION SET

- **SETB** sets either the C register or any Direct Addressed bit to one (1).
- **CPL** either forms the one's complement of the operand in the A register and returns the result to the A register without affecting flags or forms the one's complement of the C register or any Direct Addressed bit.
- **RL, RLC, RR, RRC, SWAP**. Five rotate operations can be performed on the A register; RL (rotate left), RR (rotate right), RLC (rotate left through C), RRC (rotate right through C), and SWAP (rotate left four). For RLC and RRC the C flag becomes equal to the last bit rotated out. SWAP rotates the A register left four places to exchange bits 3 through 0 with bits 7 through 4.

### Two-Operand Operations

Three two-operand logical operations are provided:

- **ANL** performs the bitwise logical conjunction of two source operands (for both bit and byte operands) and returns the result to the location of the first operand.
- **ORL** performs the bitwise logical inclusive disjunction of two source operands (for both bit and byte operands) and returns the result of the location of the first operand.
- **XRL** performs the bitwise logical exclusive disjunction of the two source operands (byte operands) and returns the result to the location of the first operand.

### 8.1.3 Arithmetic

The 8051 provides the four basic mathematical operations. Only 8-bit operations using unsigned arithmetic are supported directly. The

overflow flag permits the addition and subtraction operation to serve for both unsigned and signed binary integers. A correction operation is also provided to allow arithmetic to be performed directly on packed decimal (BCD) representations.

### Flag Register Settings

Three one-bit flag registers are set or cleared by arithmetic operations to reflect certain properties of the result of the operation. These flags are not affected by the increment and decrement instructions. A fourth flag (P) denotes the parity of the eight accumulator bits. These flag registers are located in the Program Status Word (PSW) register. Their bit assignment are shown below. A list of the instructions that affect these flags is provided in the "8051 Instruction Set Summary" in Table 8-1.

|                   |  |           |    |    |                      |     |          |   |
|-------------------|--|-----------|----|----|----------------------|-----|----------|---|
|                   |  |           |    |    |                      |     |          |   |
| SYMBOLIC ADDRESS: |  | CY        | AC | F0 | RS1                  | RS0 | OV       | P |
| REGISTER:         |  | C         |    |    |                      |     |          |   |
|                   |  | USER FLAG |    |    | REGISTER BANK SELECT |     | RESERVED |   |

| Function                         | Flag | Bit Location |
|----------------------------------|------|--------------|
| Carry Flag (also the C register) | CY:  | PSW.7:       |
| Auxiliary-Carry Flag             | AC:  | PSW.6:       |
| Overflow Flag                    | OV:  | PSW.2:       |
| Parity Flag                      | P:   | PSW.0:       |
| User Flag 0                      | F0:  | PSW.5:       |
| reserved                         | ---  | PSW.1:       |
| Register Select MSb              | RS1: | PSW.4:       |
| Register Select LSb              | RS0: | PSW.3:       |

Unless otherwise stated, the instructions obey these rules:



## MCS-51 INSTRUCTION SET

- CY is set if the operation result in a carry out of (during addition) or a borrow into (during subtraction) the high-order bit of the result; otherwise CY is cleared.
- AC is set if the operation results in a carry out of the low-order four bits of the result (during addition) or a borrow from the high-order bits into the low-order 4 bits (during subtraction); otherwise AC is cleared.
- OV is set if the operation results in a carry into the high-order bit of the result but not a carry out of the high-order bit, or vice versa; otherwise OV is cleared. OV is of use to two's-complement arithmetic, since it becomes set when the signed result cannot be represented in 8 bits.
- P is set if the module 2 sum of the eight bits in the accumulator is 1 (odd parity); otherwise P is cleared (even parity). When a value is written to the PSW register, the P bit remains unchanged, as it always reflects the parity of A.

### Addition

Four addition operations are provided:

- INC (increment) performs an addition of the source operand and one (1) and returns the result to the operand.
- ADD performs an addition between the A register and the second source operand and returns the result to the A register.
- ADDC (add with Carry) performs an addition between the A register and the second source operand; adds one (1) if the C flag is found previously set and returns the result to register A.
- DA (decimal-add-adjust for BCD addition) performs a correction to the sum

which resulted from the binary addition of two two-digit decimal operands. The packed decimal sum formed by DA is returned to A. The carry flag is set if the BCD result is greater than 99; or else it is cleared.

### Subtraction

Two subtraction operations are provided:

- SUBB (subtract with borrow) performs a subtraction of the second source operand from the first operand (the accumulator), subtracts one (1) if the C flag is found previously set and returns the result to the A register.
- DEC (decrement) performs a subtraction of one (1) from the source operand and returns the results to the operand.

### Multiplication

- MUL performs an unsigned multiplication of the A register by the B register, returning a double-byte result. Register A receives the low-order byte, B receives the high-order byte. OV is cleared if the top half of the result is zero and is set if it is non-zero. C is cleared. AC remains unaltered.

### Division

- DIV performs an unsigned division of the A register by the B register and returns the integer quotient to register A and returns the fractional remainder to the B register. Division by zero leaves indeterminate data in registers A and B and sets OV, otherwise OV is cleared. C is cleared. AC remains unaltered.



## MCS-51 INSTRUCTION SET

### 8.1.4 Control Transfer

There are three classes of control transfer operations: unconditional calls, returns and jumps; conditional jumps; and interrupts. All control transfer operations cause, some upon a specific condition, the program execution to continue at a non-sequential location in program memory.

#### Unconditional Calls, Returns and Jumps

Unconditional calls, returns and jumps transfer control from the current value of the Program Counter to the target address. Both direct and indirect transfers are supported. The three transfer operations are described below.

- **ACALL and LCALL** push the address of the next instruction onto the stack (PCL to low-order address, PCH to high-order address) and then transfer control to the target address. Absolute Call is a 2-byte instruction used when the target address is in the current 2K page. Long Call is a 3-byte instruction that addresses the full 64K program space. In ACALL, immediate data (i.e. and 11 bit address field) is concatenated to the five most significant bits of the PC (which is pointing to the next instruction). If ACALL is in the last 2 bytes of a 2K page then the call will be made to the next page since the PC will have been incremented to the next instruction prior to execution.
- **RET** transfers control to the return address saved on the stack by a previous call operation and decrements the SP register by two (2) to adjust the SP for the popped address.
- **AJMP, LJMP and SJMP** transfer control to the target operand. The operation of AJMP and LJMP are analogous to

**ACALL and LCALL.** The SJMP (short jump) instruction provides for transfers within a 256 byte range centered about the starting address of the next instruction (-128 to +127). The PC-relative short jump facilitates relocatable code.

- **JMP @ A + DPTR** performs a jump relative to the DPTR register. The operand in the A register is used as the offset (0-255) to the address in the DPTR register. Thus, the effective destination for a jump can be anywhere in the Program Memory space. This indirect jump is also useful for implementing N-way branches.

#### Conditional Jumps

In the control transfer group, the conditional jumps perform a jump contingent upon a specific condition. The destination will be within a 256-byte range centered about the starting address of the next instruction (-128 to +127).

- **JZ** performs a jump if the accumulator is zero.
- **JNZ** performs a jump if the accumulator is not zero.
- **JC** performs a jump if the carry flag is set.
- **JNC** performs a jump if the carry flag is not set.
- **JB** performs a jump if the Direct Addressed bit is set.
- **JNB** performs a jump if the Direct Addressed bit is not set.
- **JBC** performs a jump if the Direct Addressed bit is set and then clears the Direct Addressed bit.



# MCS-51 INSTRUCTION SET

**Table 8-1. 8051 Instruction Set Summary**

Interrupt Response Time: To finish execution of current instruction, respond to the interrupt request, push the PC and to vector to the first instruction of the interrupt service program requires 38 to 81 oscillator periods (3 to 7 $\mu$ s @ 12 MHz).

## INSTRUCTIONS THAT AFFECT FLAG SETTINGS\*

| INSTRUCTION | FLAG    | INSTRUCTION | FLAG    |
|-------------|---------|-------------|---------|
|             | C OV AC |             | C OV AC |
| ADD         | X X X   | CLR C       | O       |
| ADDC        | X X X   | CPL C       | X       |
| SUBB        | X X X   | ANL C,bit   | X       |
| MUL         | O X     | ANL C,/bit  | X       |
| DIV         | O X     | ORL C,bit   | X       |
| DA          | X       | ORL C,bit   | X       |
| RRC         | X       | MOV C,bit   | X       |
| RLC         | X       | CJNE        | X       |
| SETB C      | 1       |             |         |

\*Note that operations on SFR byte address 208 or bit addresses 209-215 (i.e. the PSW or bits in the PSW) will also affect flag settings.

Notes on instruction set and addressing modes:

- Rn — Register R7-R0 of the currently selected Register Bank.
- data — 8-bit internal data location's address. This could be an Internal Data RAM location (0-127) or a SFR [i.e. I/O port, control register, status register, etc. (128-255)].
- @Ri — 8-bit internal data RAM location (0-255) addressed indirectly through register R1 or R0.
- #data — 8-bit constant included in instruction.
- #data 16 — 16-bit constant included in instruction.
- addr16 — 16-bit destination address. Used by LCALL & LJMP. A branch can be anywhere within the 64K-byte Program Memory address space.
- addr11 — 11-bit destination address. Used by ACALL & AJMP. The branch will be within the same 2K-byte page of program memory as the first byte of the following instruction.
- rel — Signed (two's complement) 8-bit offset byte. Used by SJMP and all conditional jumps. Range is -128 to +127 bytes relative to first byte of the following instruction.
- bit — Direct Addressed bit in Internal Data RAM or Special Function Register.
- \*
- New operation not provided by 8048/8049.

## ARITHMETIC OPERATIONS

| Mnemonic      | Description                                | Byte | Oscillator Period |
|---------------|--|------|-------------------|
| ADD A,Rn      | Add register to Accumulator                | 1    | 12                |
| ADD A,direct  | Add direct byte to Accumulator             | 2    | 12                |
| ADD A,@Ri     | Add indirect RAM to Accumulator            | 1    | 12                |
| ADD A,#data   | Add immediate data to Accumulator          | 2    | 12                |
| ADDC A,Rn     | Add register to Accumulator with Carry     | 1    | 12                |
| ADDC A,direct | Add direct byte to Accumulator with Carry  | 2    | 12                |
| ADDC A,@Ri    | Add indirect RAM to Accumulator with Carry | 1    | 12                |
| ADDC A,#data  | Add immediate data to Acc with Carry       | 2    | 12                |
| SUBB A,Rn     | Subtract register from Acc with borrow     | 1    | 12                |

## ARITHMETIC OPERATIONS Cont.

| Mnemonic      | Description                                  | Byte | Oscillator Period |
|---------------|--|------|-------------------|
| SUBB A,direct | Subtract direct byte from Acc with borrow    | 2    | 12                |
| SUBB A,@Ri    | Subtract indirect RAM from Acc with borrow   | 1    | 12                |
| SUBB A,#data  | Subtract immediate data from Acc with borrow | 2    | 12                |
| INC A         | Increment Accumulator                        | 1    | 12                |
| INC Rn        | Increment register                           | 1    | 12                |
| INC direct    | Increment direct byte                        | 2    | 12                |
| INC @Ri       | Increment indirect RAM                       | 1    | 12                |
| DEC A         | Decrement Accumulator                        | 1    | 12                |
| DEC Rn        | Decrement Register                           | 1    | 12                |
| DEC direct    | Decrement direct byte                        | 2    | 12                |
| DEC @Ri       | Decrement indirect RAM                       | 1    | 12                |

All mnemonics copyrighted ©Intel Corporation 1980



# MCS-51 INSTRUCTION SET

**Table 8-1. Instruction Set Summary (continued)**

| ARITHMETIC OPERATIONS Cont. |                            |      |                   |  |
|-----------------------------|----------------------------|------|-------------------|--|
| Mnemonic                    | Description                | Byte | Oscillator Period |  |
| INC DPTR                    | Increment Data Pointer     | 1    | 24                |  |
| MUL AB                      | Multiply A & B             | 1    | 48                |  |
| DIV AB                      | Divide A by B              | 1    | 48                |  |
| DA A                        | Decimal Adjust Accumulator | 1    | 12                |  |

| LOGICAL OPERATIONS |  |      |                   |  |
|--------------------|--|------|-------------------|--|
| Mnemonic           | Description                                | Byte | Oscillator Period |  |
| ANL A,Rn           | AND register Accumulator                   | 1    | 12                |  |
| ANL A,direct       | AND direct byte to Accumulator             | 2    | 12                |  |
| ANL A,@Ri          | AND indirect RAM to Accumulator            | 1    | 12                |  |
| ANL A,#data        | AND immediate data to Accumulator          | 2    | 12                |  |
| ANL direct,A       | AND Accumulator to direct byte             | 2    | 12                |  |
| ANL direct,#data   | AND immediate data to direct byte          | 3    | 24                |  |
| ORL A,Rn           | OR register to Accumulator                 | 1    | 12                |  |
| ORL A,direct       | OR direct byte to Accumulator              | 2    | 12                |  |
| ORL A,@Ri          | OR indirect RAM to Accumulator             | 1    | 12                |  |
| ORL A,#data        | OR immediate data to Accumulator           | 2    | 12                |  |
| ORL direct,A       | OR Accumulator to direct byte              | 2    | 12                |  |
| ORL direct,#data   | OR immediate data to direct byte           | 3    | 24                |  |
| XRL A,Rn           | Exclusive-OR register to Accumulator       | 1    | 12                |  |
| XRL A,direct       | Exclusive-OR direct byte to Accumulator    | 2    | 12                |  |
| XRL A,@Ri          | Exclusive-OR indirect RAM to Accumulator   | 1    | 12                |  |
| XRL A,#data        | Exclusive-OR immediate data to Accumulator | 2    | 12                |  |

| LOGICAL OPERATIONS Cont. |  |      |                   |  |
|--------------------------|--|------|-------------------|--|
| Mnemonic                 | Description                                | Byte | Oscillator Period |  |
| XRL direct,A             | Exclusive-OR Accumulator to direct byte    | 2    | 12                |  |
| XRL direct,#data         | Exclusive-OR immediate data to direct byte | 3    | 24                |  |
| CLR A                    | Clear Accumulator                          | 1    | 12                |  |
| CPL A                    | Complement Accumulator                     | 1    | 12                |  |
| RL A                     | Rotate Accumulator Left                    | 1    | 12                |  |
| RLC A                    | Rotate Accumulator Left through the Carry  | 1    | 12                |  |
| RR A                     | Rotate Accumulator Right                   | 1    | 12                |  |
| RRC A                    | Rotate Accumulator Right through the Carry | 1    | 12                |  |
| SWAP A                   | Swap nibbles within the Accumulator        | 1    | 12                |  |



# MCS-51 INSTRUCTION SET

**Table 8-1. Instruction Set Summary (continued)**

| DATA TRANSFER |               |  |      |                   | DATA TRANSFER Cont. |          |  |      |                   |
|---------------|---------------|--|------|-------------------|---------------------|----------|--|------|-------------------|
|               | Mnemonic      | Description                              | Byte | Oscillator Period |                     | Mnemonic | Description                                    | Byte | Oscillator Period |
| MOV           | A,Rn          | Move register to Accumulator             | 1    | 12                | MOVX                | @Ri,A    | Move Acc to External RAM (8-bit addr)          | 1    | 24                |
| MOV           | A,direct      | Move direct byte to Accumulator          | 2    | 12                | MOVX                | @DPTR,A  | Move Acc to External Ram (16-bit addr)         | 1    | 24                |
| MOV           | A,@Ri         | Move indirect RAM to Accumulator         | 1    | 12                | PUSH                | direct   | Push direct byte onto stack                    | 2    | 24                |
| MOV           | A,#data       | Move immediate data to Accumulator       | 2    | 12                | POP                 | direct   | Pop direct byte from stack                     | 2    | 24                |
| MOV           | Rn,A          | Move Accumulator to register             | 1    | 12                | XCH                 | A,Rn     | Exchange register with Accumulator             | 1    | 12                |
| MOV           | Rn,direct     | Move direct byte to register             | 2    | 24                | XCH                 | A,direct | Exchange direct byte with Accumulator          | 2    | 12                |
| MOV           | Rn,#data      | Move immediate data to register          | 2    | 12                | XCH                 | A,@Ri    | Exchange indirect RAM with Accumulator         | 1    | 12                |
| MOV           | direct,A      | Move Accumulator to direct byte          | 2    | 12                | XCHD                | A,@Ri    | Exchange low-order Digit indirect RAM with Acc | 1    | 12                |
| MOV           | direct,Rn     | Move register to direct byte             | 2    | 24                |                     |          |  |      |                   |
| MOV           | direct,direct | Move direct byte to direct               | 3    | 24                |                     |          |  |      |                   |
| MOV           | direct,@Ri    | Move indirect RAM to direct byte         | 2    | 24                |                     |          |  |      |                   |
| MOV           | direct,#data  | Move immediate data to direct byte       | 3    | 24                |                     |          |  |      |                   |
| MOV           | @Ri,A         | Move Accumulator to indirect RAM         | 1    | 12                |                     |          |  |      |                   |
| MOV           | @Ri,direct    | Move direct byte to indirect RAM         | 2    | 24                |                     |          |  |      |                   |
| MOV           | @Ri,#data     | Move immediate data to indirect RAM      | 2    | 12                |                     |          |  |      |                   |
| MOV           | DPTR,#data16  | Load Data Pointer with a 16-bit constant | 3    | 24                |                     |          |  |      |                   |
| MOVC          | A,@A + DPTR   | Move Code byte relative to DPTR to Acc   | 1    | 24                |                     |          |  |      |                   |
| MOVC          | A,@A + PC     | Move Code byte relative to PC and Acc    | 1    | 24                |                     |          |  |      |                   |
| MOVX          | A,@Ri         | Move External RAM (8-bit addr) to Acc    | 1    | 24                |                     |          |  |      |                   |
| MOVX          | A,@DPTR       | Move External RAM (16-bit addr) to Acc   | 1    | 24                |                     |          |  |      |                   |

All mnemonics copyrighted ©Intel Corporation 1980



# MCS-51 INSTRUCTION SET

**Table 8-1. Instruction Set Summary (continued)**

| BOOLEAN VARIABLE MANIPULATION |                                       |      |                   |  |
|-------------------------------|---------------------------------------|------|-------------------|--|
| Mnemonic                      | Description                           | Byte | Oscillator Period |  |
| CLR C                         | Clear Carry                           | 1    | 12                |  |
| CLR bit                       | Clear direct bit                      | 2    | 12                |  |
| SETB C                        | Set Carry                             | 1    | 12                |  |
| SETB bit                      | Set direct bit                        | 2    | 12                |  |
| CPL C                         | Complement Carry                      | 1    | 12                |  |
| CPL bit                       | Complement direct bit                 | 2    | 12                |  |
| ANL C,bit                     | AND direct bit to Carry               | 2    | 24                |  |
| ANL C,/bit                    | AND complement of direct bit to Carry | 2    | 24                |  |
| ORL C,bit                     | OR direct bit to Carry                | 2    | 24                |  |
| ORL C,/bit                    | OR complement of direct bit to Carry  | 2    | 24                |  |
| MOV C,bit                     | Move direct bit to Carry              | 2    | 12                |  |
| MOV bit,C                     | Move Carry to direct bit              | 2    | 24                |  |
| JC rel                        | Jump if Carry is set                  | 2    | 24                |  |
| JNC rel                       | Jump if Carry not set                 | 2    | 24                |  |
| JB bit,rel                    | Jump if direct Bit is set             | 3    | 24                |  |
| JNB bit,rel                   | Jump if direct Bit is Not set         | 3    | 24                |  |
| JBC bit,rel                   | Jump if direct Bit is set & clear bit | 3    | 24                |  |

| PROGRAMMING BRANCHING |                                    |      |                   |  |
|-----------------------|------------------------------------|------|-------------------|--|
| Mnemonic              | Description                        | Byte | Oscillator Period |  |
| ACALL addr11          | Absolute Subroutine Call           | 2    | 24                |  |
| LCALL addr16          | Long Subroutine Call               | 3    | 24                |  |
| RET                   | Return for Subroutine              | 1    | 24                |  |
| RETI                  | Return for interrupt               | 1    | 24                |  |
| AJMP addr11           | Absolute Jump                      | 2    | 24                |  |
| LJMP addr16           | Long Jump                          | 3    | 24                |  |
| SJMP rel              | Short Jump (relative addr)         | 2    | 24                |  |
| JMP @A + DPTR         | Jump indirect relative to the DPTR | 1    | 24                |  |
| JZ rel                | Jump if Accumulator is Zero        | 2    | 24                |  |

| PROGRAM BRANCHING Cont. |   |      |                   |  |
|-------------------------|---|------|-------------------|--|
| Mnemonic                | Description   | Byte | Oscillator Period |  |
| JNZ rel                 | Jump if Accumulator is Not Zero                     | 2    | 24                |  |
| CJNE A,direct,rel       | Compare direct byte to Acc and Jump if Not Equal    | 3    | 24                |  |
| CJNE A,#data,rel        | Compare immediate to Acc and Jump if Not Equal      | 3    | 24                |  |
| CJNE RN,#data,rel       | Compare immediate to register and Jump if Not Equal | 3    | 24                |  |
| CJNE @Ri,#data,rel      | Compare immediate to indirect and Jump if Not Equal | 3    | 24                |  |
| DJNZ Rn,rel             | Decrement register and Jump if Not Zero             | 3    | 24                |  |
| DJNZ direct,rel         | Decrement direct byte and Jump if Not Zero          | 3    | 24                |  |
| NOP                     | No Operation  | 1    | 12                |  |

Register Addressing may be used to address the source operand.

Program execution control may be transferred by means of internal and external interrupts. All interrupts perform a transfer by pushing the Program Counter onto the stack and then branching to programs located at absolute locations 3, 11, 19, 27 and 35 in the Program Memory. The programmer must push all registers that will be altered by his interrupt service program onto the stack to avoid corruption. Only one interrupt transfer operation is necessary:

\* RETI transfers control in a manner identical to RET. In addition, RETI re-enables interrupts for the current priority level.

All mnemonics copyrighted ©Intel Corporation 1980



## MCS-51 INSTRUCTION SET

- CJNE compares the first operand to the second operand and performs a jump if they are not equal. C is set if the first operand is less than the second operand; else it is cleared. Comparisons can be made between the A register and Direct Addressable bytes in the Internal Data Memory or between an immediate value and either the A register, an RB register in the selected Register Bank, or a Register-Indirect addressed byte of the Internal Data RAM.
- DJNZ decrements the source operand and returns the result to the operand. A jump is performed if the result is not zero. The DJNZ instruction makes a RAM location efficient for use as a program loop counter by allowing the programmer to decrement and test the counter in a single instruction. The source operand of the DJNZ instruction may be any byte in the Internal Data Memory. Either Direct or Register Addressing may be used to address the source operand.

### Interrupts

Program execution control may be transferred by means of internal and external interrupts. All interrupts perform a transfer by pushing the Program Counter onto the stack and then branching to programs located at absolute locations 3, 11, 19, 27 and 35 in the Program Memory. The programmer must push all registers that will be altered by his interrupt service program onto the stack to avoid corruption. Only one interrupt transfer operation is necessary:

- RETI transfers control in a manner identical to RET. In addition, RETI reenables interrupts for the current priority level.

See section 2.3.1 for further details on the operation and control of the interrupt system.

### 8.2 Instruction Definitions

The rest of this chapter defines all the instructions and operations which the MCS-51 CPU can perform. There is a separate section for each of the 51 basic operations, ordered alphabetically according to the operation mnemonic.

When an operation may apply to more than one data type (generally bit and byte data), the MCS-51 assembly language uses the same mnemonic for each, reducing the number of mnemonics the programmer must remember. The assembler determines which instruction is appropriate from the operands specified. Thus, the mnemonic "CLR" can operate on the eight-bit accumulator ("CLR A"), or on one-bit variables ("CLR F0"). The mnemonics ANL, ORL, CPL, and MOV can relate to more than one data type as well. These operations present each data type in a separate section.

Each section then describes the action taken by the operation, the flags and registers affected, and shows a short example of how an instruction might be used in a program. Next comes the number of bytes and machine cycles required, the corresponding binary machine-language encoding, and a symbolic description or restatement of the function implemented.

*Note:* Only the carry, auxiliary-carry, and overflow flags are discussed in these instruction descriptions. Since the parity bit (PSW.0) is recomputed *after every instruction cycle* any instruction that alters the accumulator — either inherently or as a special function



MCS-51 INSTRUCTION SET

register — could affect the parity flag. Similarly, instructions which alter directly addressed registers could affect the other status flags if the instruction is applied to the PSW. Status flags can also be modified by the generalized bit-manipulation instructions.

Nineteen operations allow more than one addressing mode for the source and/or destination operand. The headings for these sections show the instruction format with such operands enclosed in angle brackets (for example, MOV <dest-byte> , <src-byte> ). The

operation description tells what modes (or combinations of modes) are allowed, and gives the assembly language notation, byte and cycle counts, encoding format, and a symbolic description for each.

The information in this chapter is directed towards defining the capabilities of the MCS-51 architecture and hardware. For details on the assembly language or ASM51 capabilities refer to the *MCS-51 Macro Assembler User's Guide*, publication number 9800937.

ACALL      addr11

**Function:** Absolute Call

**Description:** ACALL unconditionally calls a subroutine located at the indicated address. The instruction increments the PC twice to obtain the address of the following instruction, then pushes the 16-bit result onto the stack (low-order byte first) and increments the stack pointer twice. The destination address is obtained by successively concatenating the five high-order bits of the incremented PC, opcode bits 7-5, and the second byte of the instruction. The subroutine called must therefore start within the same 2K block of the program memory as the first byte of the instruction following ACALL. No flags are affected.

**Example:** Initially SP equals 07H. The label "SUBRTN" is at program memory location 0345H. After executing the instruction,

ACALL SUBRTN

at location 0123H, SP will contain 09H, internal RAM locations 08H and 09H will contain 25H and 01H, respectively, and the PC will contain 0345H.

**Bytes:** 2  
**Cycles:** 2

|           |                     |  |  |  |  |  |  |  |                         |  |  |  |  |  |  |  |
|-----------|---------------------|--|--|--|--|--|--|--|-------------------------|--|--|--|--|--|--|--|
| Encoding: | a10 a9 a8 1 0 0 0 1 |  |  |  |  |  |  |  | a7 a6 a5 a4 a3 a2 a1 a0 |  |  |  |  |  |  |  |
|           |                     |  |  |  |  |  |  |  |                         |  |  |  |  |  |  |  |

**Operation:** ACALL  
(PC) ← (PC) + 2  
(SP) ← (SP) + 1  
((SP)) ← (PC7-0)  
(SP) ← (SP) + 1  
((SP)) ← (PC15-8)  
(PC10-0) ← page address



## MCS-51 INSTRUCTION SET

### ADD A,<src-byte>

**Function:** Add

**Description:** ADD adds the byte variable indicated to the accumulator, leaving the result in the accumulator. The carry and auxiliary-carry flags are set, respectively, if there is a carry-out from bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occurred.

OV is set if there is a carry-out of bit 6 but not out of bit 7, or a carry-out of bit 7 but not bit 6; otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands, or a positive sum from two negative operands.

Four source operand addressing modes are allowed: register, direct, register-indirect, or immediate.

**Example:** The accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B). The instruction,

ADD A,R0

will leave 6DH (01101101B) in the accumulator with the AC flag cleared and both the carry flag and OV set to 1.

#### ADD A,Rn

**Bytes:** 1

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 1 | r | r | r |
|---|---|---|---|---|---|---|---|

**Operation:**

ADD

$(A) \leftarrow (A) + (Rn)$

#### ADD A,direct

**Bytes:** 2

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|

direct address

**Operation:**

ADD

$(A) \leftarrow (A) + (\text{direct})$



## MCS-51 INSTRUCTION SET

### ADD A,@Ri

Bytes: 1

Cycles: 1

Encoding:

|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | i |
|---|---|---|---|---|---|---|---|---|

Operation:

ADD

$(A) \leftarrow (A) + ((Ri))$

### ADD A,#data

Bytes: 2

Cycles: 1

Encoding:

|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|

immediate data

Operation:

ADD

$(A) \leftarrow (A) + \#data$

C3  
AA  
6



## MCS-51 INSTRUCTION SET

### ADDC    A, <src-byte>

**Function:** Add with Carry  
**Description:** ADDC simultaneously adds the byte variable indicated, the carry flag and the accumulator contents, leaving the result in the accumulator. The carry and auxiliary-carry flags are set, respectively, if there is a carry-out from bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occurred.

OV is set if there is a carry-out of bit 6 but not out of bit 7, or a carryout of bit 7 but not out of bit 6; otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands or a positive sum from two negative operands.

Four source operand addressing modes are allowed: register, direct, register-indirect, or immediate.

**Example:** The accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B) with the carry flag set. The instruction,

ADDC    A,R0

will leave 6EH (01101110B) in the accumulator with AC cleared and both the carry flag and OV set to 1.

### ADDC   A,Rn

**Bytes:** 1

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 1 | r | r | r |
|---|---|---|---|---|---|---|---|

**Operation:**    ADDC  
                    $(A) \leftarrow (A) + (C) + (R_n)$

### ADDC   A,direct

**Bytes:** 2

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|

|                |
|----------------|
| direct address |
|----------------|

**Operation:**    ADDC  
                    $(A) \leftarrow (A) + (C) + (\text{direct})$



## MCS-51 INSTRUCTION SET

### ADDC A,@Ri

**Bytes:** 1

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 | 1 | 1 | i |
|---|---|---|---|---|---|---|---|

**Operation:** ADDC

$$(A) \leftarrow (A) + (C) + ((R_i))$$

### ADDC A,#data

**Bytes:** 2

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |                |
|---|---|---|---|---|---|---|---|----------------|
| 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | immediate data |
|---|---|---|---|---|---|---|---|----------------|

**Operation:** ADDC

$$(A) \leftarrow (A) + (C) + \#data$$

### AJMP addr11

**Function:** Absolute Jump

**Description:** AJMP transfers program execution to the indicated address, which is formed at run-time by concatenating the high-order five bits of the PC (*after* incrementing the PC twice), opcode bits 7-5, and the second byte of the instruction. The destination must therefore be within the same 2K block of program memory as the first byte of the instruction following AJMP.

**Example:** The label "JMPADR" is at program memory location 0123H. The instruction, AJMP JMPADR

is at location 0345H and will load the PC with 0123H.

**Bytes:** 2

**Cycles:** 2

**Encoding:**

|     |    |    |   |   |   |   |   |    |    |    |    |    |    |    |    |
|-----|----|----|---|---|---|---|---|----|----|----|----|----|----|----|----|
| a10 | a9 | a8 | 0 | 0 | 0 | 0 | 1 | a7 | a6 | a5 | a4 | a3 | a2 | a1 | a0 |
|-----|----|----|---|---|---|---|---|----|----|----|----|----|----|----|----|

**Operation:** AJMP

$$(PC) \leftarrow (PC) + 2$$

$$(PC_{10-0}) \leftarrow \text{page address}$$



## MCS-51 INSTRUCTION SET

**ANL**     <dest-byte> , <src-byte>

**Function:** Logical-AND for byte variables  
**Description:** ANL performs the bitwise logical-AND operation between the variables indicated and stores the results in the destination variable. No flags are affected.

The two operands allow six addressing mode combinations. When the destination is the accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the accumulator or immediate data.

*Note:* When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

**Example:** If the accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B) then the instruction,

ANL     A,R0

will leave 41H (01000001B) in the accumulator.

When the destination is a directly addressed byte, this instruction will clear combinations of bits in any RAM location or hardware register. The mask byte determining the pattern of bits to be cleared would either be a constant contained in the instruction or a value computed in the accumulator at run-time. The instruction,

ANL     P1,#01110011B

will clear bits 7, 3, and 2 of output port 1.

**ANL   A,Rn**

**Bytes:**     1

**Cycles:**    1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 1 | r | r | r |
|---|---|---|---|---|---|---|---|

**Operation:**     ANL  
                    $(A) \leftarrow (A) \wedge (Rn)$



## MCS-51 INSTRUCTION SET

### ANL A,direct

Bytes: 2

Cycles: 1

Encoding: 

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|

direct address

Operation: ANL  
 $(A) \leftarrow (A) \wedge (\text{direct})$

### ANL A,@Ri

Bytes: 1

Cycles: 1

Encoding: 

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 0 | 1 | 1 | i |
|---|---|---|---|---|---|---|---|

Operation: ANL  
 $(A) \leftarrow (A) \wedge ((Ri))$

### ANL A,#data

Bytes: 2

Cycles: 1

Encoding: 

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
|---|---|---|---|---|---|---|---|

immediate data

Operation: ANL  
 $(A) \leftarrow (A) \wedge \#data$

### ANL direct,A

Bytes: 2

Cycles: 1

Encoding: 

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
|---|---|---|---|---|---|---|---|

direct address

Operation: ANL  
 $(\text{direct}) \leftarrow (\text{direct}) \wedge (A)$

### ANL direct,#data

Bytes: 3

Cycles: 2

Encoding: 

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|

direct address immediate data

Operation: ANL  
 $(\text{direct}) \leftarrow (\text{direct}) \wedge \#data$



## MCS-51 INSTRUCTION SET

### ANL C, <src-bit>

**Function:** Logical-AND for bit variables

**Description:** If the Boolean value of the source bit is a logical 0 then clear the carry flag; otherwise leave the carry flag in its current state. A slash ("/") preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, *but the source bit itself is not affected*. No other flags are affected.

**Example:** Only direct bit addressing is allowed for the source operand.  
Set the carry flag if, and only if, P1.0 = 1, ACC. 7 = 1, and OV = 0:

```
MOV  C,P1.0           ;LOAD CARRY WITH INPUT PIN STATE
ANL  C,ACC.7          ;AND CARRY WITH ACCUM. BIT 7
ANL  C,/OV             ;AND WITH INVERSE OF OVERFLOW
                        FLAG
```

#### ANL C,bit

**Bytes:** 2

**Cycles:** 2

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
|---|---|---|---|---|---|---|---|

bit address

**Operation:** ANL  
 $(C) \leftarrow (C) \wedge (\text{bit})$

#### ANL C,/bit

**Bytes:** 2

**Cycles:** 2

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|

bit address

**Operation:** ANL  
 $(C) \leftarrow (C) \neg (\text{bit})$

### CJNE <dest-byte>,<src-byte>, rel

**Function:** Compare and Jump if Not Equal.

**Description:** CJNE compares the magnitudes of the first two operands, and branches if their values are not equal. The branch destination is computed by adding the signed relative-displacement in the last instruction byte to the PC, after incrementing the PC to the start of the next instruction. The carry flag is set if the unsigned integer value of <dest-byte> is less than the unsigned integer value of <src-byte>; otherwise, the carry is cleared. Neither operand is affected.



## MCS-51 INSTRUCTION SET

The first two operands allow four addressing mode combinations: the accumulator may be compared with any directly addressed byte or immediate data, and any indirect RAM location or working register can be compared with an immediate constant.

**Example:** The accumulator contains 34H. Register 7 contains 56H. The first instruction in the sequence,

```

                CJNE    R7,#60H, NOT_EQ
;
; NOT_EQ:    ...      ...      ; R7 = 60H.
;            JC      REQ_LOW    ; IF R7 < 60H.
;            ...      ...      ; R7 > 60H.

```

sets the carry flag and branches to the instruction at label NOT\_EQ. By testing the carry flag, this instruction determines whether R7 is greater or less than 60H.

If the data being presented to port 1 is also 34H, then the instruction,

```
WAIT: CJNE A,P1,WAIT
```

clears the carry flag and continues with the next instruction in sequence, since the accumulator does equal the data read from P1. (If some other value was being input on P1, the program will loop at this point until the P1 data changes to 34H.)

**CJNE A,direct,rel**

**Bytes:** 3

**Cycles:** 2

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|

direct address

rel. address

**Operation:**

CJNE

$(PC) \leftarrow (PC) + 3$

IF (direct) < (A)

THEN  $(PC) \leftarrow (PC) + \text{rel and } (C) \leftarrow 0$

OR

IF (direct) < (A)

THEN  $(PC) \leftarrow (PC) + \text{rel and } (C) \leftarrow 1$



## MCS-51 INSTRUCTION SET

### CJNE A,#data,rel

Bytes: 3

Cycles: 2

Encoding: 

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |
|---|---|---|---|---|---|---|---|

Operation: CJNE

$(PC) \leftarrow (PC) + 3$

IF #data < (A)

THEN  $(PC) \leftarrow (PC) + \text{rel}$  and  $(C) \leftarrow 0$

OR

IF #data > (A)

THEN  $(PC) \leftarrow (PC) + \text{rel}$  and  $(C) \leftarrow 1$

### CJNE Rn,#data,rel

Bytes: 3

Cycles: 2

Encoding: 

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 1 | r | r | r |
|---|---|---|---|---|---|---|---|

Operation: CJNE

$(PC) \leftarrow (PC) + 3$

IF #data < (Rn)

THEN  $(PC) \leftarrow (PC) + \text{rel}$  and  $(C) \leftarrow 0$

OR

IF #data > (Rn)

THEN  $(PC) \leftarrow (PC) + \text{rel}$  and  $(C) \leftarrow 1$

### CJNE @Ri,#data,rel

Bytes: 3

Cycles: 2

Encoding: 

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 | 1 | 1 | i |
|---|---|---|---|---|---|---|---|

Operation: CJNE

$(PC) \leftarrow (PC) + 3$

IF #data < ((Ri))

THEN  $(PC) \leftarrow (PC) + \text{rel}$  and  $(C) \leftarrow 1$

OR

IF #data > ((Ri))

THEN  $(PC) \leftarrow (PC) + \text{rel}$  and  $(C) \leftarrow 0$



## MCS-51 INSTRUCTION SET

### CLR A

**Function:** Clear Accumulator

**Description:** The accumulator is cleared (all bits set to zero). No flags are affected.

**Example:** The accumulator contains 5CH (01011100B). The instruction,

CLR A

will leave the accumulator set to 00H (00000000B).

**Bytes:** 1

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
|---|---|---|---|---|---|---|---|

**Operation:**

CLR  
(A) ← 0

### CLR bit

**Function:** Clear bit

**Description:** The indicated bit is cleared (reset to zero). No other flags are affected. CLR can operate on the carry flag or any directly addressable bit.

**Example:** Port 1 has previously been written with 5DH (01011101B). The instruction,

CLR P1.2

will leave the port set to 59H (01011001B).

### CLR C

**Bytes:** 1

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|

**Operation:**

CLR  
(C) ← 0

### CLR bit

**Bytes:** 2

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
|---|---|---|---|---|---|---|---|

bit address

**Operation:**

CLR  
(bit) ← 0



## MCS-51 INSTRUCTION SET

### CPL A

**Function:** Complement Accumulator

**Description:** Each bit of the accumulator is logically complemented (one's complement). Bits which previously contained a one are changed to zero and vice-versa. No flags are affected.

**Example:** The accumulator contains 5CH (01011100B). The instruction,

CPL A

will leave the accumulator set to 0A3H (10100011B).

**Bytes:** 1

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |
|---|---|---|---|---|---|---|---|

**Operation:**

CPL  
 $(A) \leftarrow \neg (A)$

### CPL bit

**Function:** Complement bit

**Description:** The bit variable specified is complemented. A bit which had been a one is changed to zero and vice-versa. No other flags are affected. CLR can operate on the carry or any directly addressable bit.

**Example:** *Note:* When this instruction is used to modify an output pin, the value used as the original data will be read from the output data latch, *not* the input pin. Port 1 has previously been written with 5BH (01011101B). The instruction sequence,

CPL P1.1

CPL P1.2

will leave the port set to 5BH (01011011B).

### CPL C

**Bytes:** 1

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|

**Operation:**

CPL  
 $(C) \leftarrow \neg (C)$

## MCS-51 INSTRUCTION SET

**CPL** bit

**Bytes:** 2

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
|---|---|---|---|---|---|---|---|

bit address

**Operation:** CPL

(bit) ←  $\neg$ (bit)

**DA A**

**Function:** Decimal-adjust Accumulator for Addition

**Description:** DA A adjusts the eight-bit value in the accumulator resulting from the earlier addition of two variables (each in packed-BCD format), producing two four-bit digits. Any ADD or ADDC instruction may have been used to perform the addition.

If accumulator bits 3-0 are greater than nine (xxxx1010-xxxx1111), or if the AC flag is one, six is added to the accumulator producing the proper BCD digit in the low-order nibble. This internal addition would set the carry flag if a carry-out of the low-order four-bit field propagated through all high-order bits, but it would not clear the carry flag otherwise.

If the carry flag is now set, or if the four high-order bits now exceed nine (1010xxxx-1111xxxx), these high-order bits are incremented by six, producing the proper BCD digit in the high-order nibble. Again, this would set the carry flag if there was a carry-out of the high-order bits, but wouldn't clear the carry. The carry flag thus indicates if the sum of the original two BCD variables is greater than 100, allowing multiple precision decimal addition. OV is not affected.

All of this occurs during the one instruction cycle. Essentially, this instruction performs the decimal conversion by adding 00H, 06H, 60H, or 66H to the accumulator, depending on initial accumulator and PSW conditions.

*Note:* DA A cannot simply convert a hexadecimal number in the accumulator to BCD notation, nor does DA A apply to decimal subtraction.



## MCS-51 INSTRUCTION SET

**Example:** The accumulator holds the value 56H (01010110B) representing the packed BCD digits of the decimal number 56. Register 3 contains the value 67H (01100111B) representing the packed BCD digits of the decimal number 67. The carry flag is set. The instruction sequence,

```
ADDC  A,R3
DA     A
```

will first perform a standard two's-complement binary addition, resulting in the value 0BEH (10111110) in the accumulator. The carry and auxiliary carry flags will be cleared.

The Decimal Adjust instruction will then alter the accumulator to the value 24H (00100100B), indicating the packed BCD digits of the decimal number 24, the low-order two digits of the decimal sum of 56, 67, and the carry-in. The carry flag will be set by the Decimal Adjust instruction, indicating that a decimal overflow occurred. The true sum 56, 67, and 1 is 124.

BCD variables can be incremented or decremented by adding 01H or 99H. If the accumulator initially holds 30H (representing the digits of 30 decimal), then the instruction sequence,

```
ADD  A,#99H
DA   A
```

will leave the carry set and 29H in the accumulator, since  $30 + 99 = 129$ . The low-order byte of the sum can be interpreted to mean  $30 + 1 = 29$ .

**Bytes:**

**Cycles:**

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
|---|---|---|---|---|---|---|---|

**Operation:**

DA

-contents of Accumulator are BCD

IF  $[(A3-0) > 9] \vee [(C) = 1]$

THEN  $(A3-0) \leftarrow (A3-0) + 6$

AND

IF  $[(A7-4) > 9] \vee [(C) = 1]$

THEN  $(A7-4) \leftarrow (A7-4) + 6$

## MCS-51 INSTRUCTION SET

### DEC     byte

**Function:** Decrement

**Description:** The variable indicated is decremented by 1. An original value of 00H will underflow to 0FFH. No flags are affected. Four operand addressing modes are allowed: accumulator, register, direct, or register-indirect.

*Note:* When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

**Example:** Register 0 contains 7FH (01111111B). Internal RAM locations 7EH and 7FH contain 00H and 40H, respectively. The instruction sequence,

```
DEC @R0
DEC R0
DEC @R0
```

will leave register 0 set to 7EH and internal RAM locations 7EH and 7FH set to 0FFH and 3FH.

### DEC A

**Bytes:** 1

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
|---|---|---|---|---|---|---|---|

**Operation:** DEC

$(A) \leftarrow (A) - 1$

### DEC Rn

**Bytes:** 1

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | r | r | r |
|---|---|---|---|---|---|---|---|

**Operation:** DEC

$(Rn) \leftarrow (Rn) - 1$



## MCS-51 INSTRUCTION SET

### DEC direct

**Bytes:** 2

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |                |
|---|---|---|---|---|---|---|---|----------------|
| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | direct address |
|---|---|---|---|---|---|---|---|----------------|

**Operation:**

DEC

$(\text{direct}) \leftarrow (\text{direct}) - 1$

### DEC @Ri

**Bytes:** 1

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 1 | 1 | i |
|---|---|---|---|---|---|---|---|

**Operation:**

DEC

$((Ri)) \leftarrow ((Ri)) - 1$

### DIV AB

**Function:**

Divide

**Description:**

DIV AB divides the unsigned eight-bit integer in the accumulator by the unsigned eight-bit integer in register B. The accumulator receives the integer part of the quotient; register B receives the integer remainder. The carry and OV flags will be cleared.

*Exception:* if B had originally contained 00H, the values returned in the accumulator and B-register will be undefined and the overflow flag will be set. The carry flag is cleared in any case.

**Example:**

The accumulator contains 251 (0FBH or 11111011B) and B contains 18 (12H or 00010010B). The instruction,

DIV AB

will leave 13 in the accumulator (0DH or 00001101B) and the value 17 (11H or 00010001B) in B, since  $251 = (13 \times 18) + 17$ . Carry and OV will both be cleared.

**Bytes:** 1

**Cycles:** 4

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
|---|---|---|---|---|---|---|---|

**Operation:**

DIV

$(A)_{15-8} \leftarrow (A) / (B)$   
 $(B)_{7-0}$

## MCS-51 INSTRUCTION SET

**DJNZ**     <byte>, <rel-addr>

**Function:** Decrement and Jump if Not Zero

**Description:** DJNZ decrements the location indicated by 1, and branches to the address indicated by the second operand if the resulting value is not zero. An original value of 00H will underflow to 0FFH. No flags are affected. The branch destination would be computed by adding the signed relative-displacement value in the last instruction byte to the PC, after incrementing the PC to the first byte of the following instruction.

The location decremented may be a register or directly addressed byte.

*Note:* When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

**Example:** Internal RAM locations 40H, 50H, and 60H contain the values 01H, 70H, and 15H, respectively. The instruction sequence,

```
DJNZ 40H, LABEL_1
DJNZ 50H, LABEL_2
DJNZ 60H, LABEL_3
```

will cause a jump to the instruction at label LABEL\_2 with the values 00H, 6FH, and 15H in the three RAM locations. The first jump was *not* taken because the result was zero.

This instruction provides a simple way of executing a program loop a given number of times, or for adding a moderate time delay (from 2 to 512 machine cycles) with a single instruction. The instruction sequence,

```
MOV R2, #8
TOGGLE: CPL P1.7
        DJNZ R2, TOGGLE
```

will toggle P1.7 eight times, causing four output pulses to appear at bit 7 of output port 1. Each pulse will last three machine cycles; two for DJNZ and one to alter the pin.



## MCS-51 INSTRUCTION SET

### DJNZ Rn,rel

**Bytes:** 2

**Cycles:** 2

**Encoding:**

|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 1 | r | r | r | r |
|---|---|---|---|---|---|---|---|---|

direct address

**Operation:**

DJNZ  
 $(PC) \leftarrow (PC) + 2$   
 $(Rn) \leftarrow (Rn) - 1$   
 IF  $(Rn) > 0$  or  $(Rn) < 0$   
 THEN  
 $(PC) \leftarrow (PC) + rel$

### DJNZ direct,rel

**Bytes:** 3

**Cycles:** 2

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|

direct address
rel. address

**Operation:**

DJNZ  
 $(PC) \leftarrow (PC) + 2$   
 $(direct) \leftarrow (direct) - 1$   
 IF  $(direct) > 0$  or  $(direct) < 0$   
 THEN  
 $(PC) \leftarrow (PC) + rel$

### INC <byte>

**Function:** Increment

**Description:** INC increments the indicated variable by 1. An original value of 0FFH will overflow to 00H. No flags are affected. Three addressing modes are allowed: register, direct, or register-indirect.

*Note:* When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

## MCS-51 INSTRUCTION SET

**Example:** Register 0 contains 7EH (01111110B). Internal RAM locations 7EH and 7FH contain 0FFH and 40H, respectively. The instruction sequence,

```
INC @R0
INC R0
INC @R0
```

will leave register 0 set to 7FH and internal RAM locations 7EH and 7FH holding (respectively) 00H and 41H.

**INC A**

**Bytes:** 1

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
|---|---|---|---|---|---|---|---|

**Operation:** INC  
 $(A) \leftarrow (A) + 1$

**INC Rn**

**Bytes:** 1

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | r | r | r |
|---|---|---|---|---|---|---|---|

**Operation:** INC  
 $(Rn) \leftarrow (Rn) + 1$

**INC direct**

**Bytes:** 2

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|---|

direct address

**Operation:** INC  
 $(\text{direct}) \leftarrow (\text{direct}) + 1$



## MCS-51 INSTRUCTION SET

### INC @Ri

**Bytes:** 1

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | i |
|---|---|---|---|---|---|---|---|---|

**Operation:**

INC  
 $((Ri)) \leftarrow ((Ri)) + 1$

### INC DPTR

**Function:**

Increment Data Pointer

**Description:**

Increment the 16-bit data pointer by 1. A 16-bit increment (modulo 2<sup>16</sup>) is performed; an overflow of the low-order byte of the data pointer (DPL) from 0FFH to 00H will increment the high-order byte (DPH). No flags are affected.

**Example:**

This is the only 16-bit register which can be incremented. Registers DPH and DPL contain 12H and 0FEH, respectively. The instruction sequence,

```
INC DPTR
INC DPTR
INC DPTR
```

will change DPH and DPL to 13H and 01H.

**Bytes:** 1

**Cycles:** 2

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|

**Operation:**

INC  
 $(DPTR) \leftarrow (DPTR) + 1$

## MCS-51 INSTRUCTION SET

### JB      bit,rel

**Function:** Jump if Bit set

**Description:** If the indicated bit is a one, jump to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. *The bit tested is not modified.* No flags are affected.

**Example:** The data present at input port 1 is 11001010B. The accumulator holds 56 (01010110B). The instruction sequence,

JB      P1.2,LABEL1

JB      ACC.2,LABEL2

will cause program execution to branch to the instruction at label LABEL2.

**Bytes:** 3

**Cycles:** 2

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|

|             |
|-------------|
| bit address |
|-------------|

|              |
|--------------|
| rel. address |
|--------------|

**Operation:**

JB

$(PC) \leftarrow (PC) + 3$

IF (bit) = 1

THEN

$(PC) \leftarrow (PC) + \text{rel}$

### JBC      bit,rel

**Function:** Jump if Bit is set and Clear bit

**Description:** If the indicated bit is one, branch to the address indicated; otherwise proceed with the next instruction. *In either case, clear the designated bit.* The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. No flags are affected.

*Note:* When this instruction is used to test an output pin, the value used as the original data will be read from the output data latch, *not* the input pin.

**Example:** The accumulator holds 56H (01010110B). The instruction sequence,

JBC      ACC.3,LABEL1

JBC      ACC.2,LABEL2

will cause program execution to continue at the instruction identified by the label LABEL2, with the accumulator modified to 52H (01010010B).



## MCS-51 INSTRUCTION SET

**Bytes:** 3

**Cycles:** 2

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|

**Operation:** JBC  
 $(PC) \leftarrow (PC) + 3$   
 IF (bit) = 1  
 THEN  
     (bit)  $\leftarrow$  0  
      $(PC) \leftarrow (PC) + \text{rel}$

**JC**      **rel**

**Function:** Jump if Carry is set  
**Description:** If the carry flag is set, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. No flags are affected.

**Example:** The carry flag is cleared. The instruction sequence,

```
JC    LABEL1
CPL   C
JC    LABEL2
```

will set the carry and cause program execution to continue at the instruction identified by the label LABEL2.

**Bytes:** 2

**Cycles:** 2

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|

**Operation:** JC  
 $(PC) \leftarrow (PC) + 2$   
 IF (C) = 1  
 THEN  
      $(PC) \leftarrow (PC) + \text{rel}$

## MCS-51 INSTRUCTION SET

### JMP @A + DPTR

**Function:** Jump indirect

**Description:** Add the eight-bit unsigned contents of the accumulator with the sixteen-bit data pointer, and load the resulting sum to the program counter. This will be the address for subsequent instruction fetches. Sixteen-bit addition is performed (modulo 2<sup>16</sup>): a carry-out from the low-order eight bits propagates through the higher-order bits. Neither the accumulator nor the data pointer is altered. No flags are affected.

**Example:** An even number from 0 to 6 is in the accumulator. The following sequence of instructions will branch to one of four AJMP instructions in a jump table starting at JMP\_\_TBL:

```

MOV     DPTR,#JMP__TBL
JMP     @A + DPTR
JMP__TBL: AJMP  LABEL0
          AJMP  LABEL1
          AJMP  LABEL2
          AJMP  LABEL3
    
```

If the accumulator equals 04H when starting this sequence, execution will jump to label LABEL2. Remember that AJMP is a two-byte instruction, so the jump instructions start at every other address.

**Bytes:** 1

**Cycles:** 2

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|

**Operation:**  $(PC) \leftarrow (A) + (DPTR)$



## MCS-51 INSTRUCTION SET

### JNB bit,rel

**Function:** Jump if Bit Not set

**Description:** If the indicated bit is a zero, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. *The bit tested is not modified.* No flags are affected.

**Example:** The data present at input port 1 is 11001010B. The accumulator holds 56H (01010110B). The instruction sequence,

```
JNB P1.3,LABEL1
JNB ACC.3,LABEL2
```

will cause program execution to continue at the instruction at label LABEL2.

**Bytes:** 3  
**Cycles:** 2

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|

|             |
|-------------|
| bit address |
|-------------|

|              |
|--------------|
| rel. address |
|--------------|

**Operation:** JNB  
 $(PC) \leftarrow (PC) + 3$   
 IF (bit) = 0  
     THEN  $(PC) \leftarrow (PC) + \text{rel.}$

### JNC rel

**Function:** Jump if Carry not set

**Description:** If the carry flag is a zero, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice to point to the next instruction. The carry flag is not modified.

**Example:** The carry flag is set. The instruction sequence,

```
JNC LABEL1
CPL C
JNC LABEL2
```

will clear the carry and cause program execution to continue at the instruction identified by the label LABEL2.

## MCS-51 INSTRUCTION SET

**Bytes:** 2  
**Cycles:** 2

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|

|              |
|--------------|
| rel. address |
|--------------|

**Operation:** JNC  
 $(PC) \leftarrow (PC) + 2$   
IF  $(C) = 0$   
THEN  $(PC) \leftarrow (PC) + \text{rel}$

**JNZ rel**

**Function:** Jump if accumulator Not Zero

**Description:** If any bit of the accumulator is a one, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. The accumulator is not modified. No flags are affected.

**Example:** The accumulator originally holds 00H. The instruction sequence,

```
JNZ LABEL1
INC A
JNZ LABEL2
```

will set the accumulator to 01H and continue at label LABEL2.

**Bytes:** 2  
**Cycles:** 2

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|

|              |
|--------------|
| rel. address |
|--------------|

**Operation:** JNZ  
 $(PC) \leftarrow (PC) + 2$   
IF  $(A) \neq 0$   
THEN  $(PC) \leftarrow (PC) + \text{rel}$



## MCS-51 INSTRUCTION SET

**JZ**      **rel**

**Function:** Jump if Accumulator Zero

**Description:** If all bits of the accumulator are zero, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. The accumulator is not modified. No flags are affected.

**Example:** The accumulator originally contains 01H. The instruction sequence,

```
JZ    LABEL1
DEC   A
JZ    LABEL2
```

will change the accumulator to 00H and cause program execution to continue at the instruction identified by the label LABEL2.

**Bytes:** 2

**Cycles:** 2

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|

rel. address

**Operation:** JZ  
 $(PC) \leftarrow (PC) + 2$   
 IF  $(A) = 0$   
     THEN  $(PC) \leftarrow (PC) + rel$

**LCALL**      **addr16**

**Function:** Long Call

**Description:** LCALL calls a subroutine located at the indicated address. The instruction adds three to the program counter to generate the address of the next instruction and then pushes the 16-bit result onto the stack (low byte first), incrementing the stack pointer by two. The high-order and low-order bytes of the PC are then loaded, respectively, with the second and third bytes of the LCALL instruction. Program execution continues with the instruction at this address. The subroutine may therefore begin anywhere in the full 64K-byte program memory address space. No flags are affected.

## MCS-51 INSTRUCTION SET

**Example:** Initially the stack pointer equals 07H. The label "SUBRTN" is assigned to program memory location 1234H. After executing the instruction,

**LCALL SUBRTN**

at location 0123H, the stack pointer will contain 09H, internal RAM locations 08H and 09H will contain 26H and 01H, and the PC will contain 1235H.

**Bytes:** 3

**Cycles:** 2

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
|---|---|---|---|---|---|---|---|

|                |
|----------------|
| addr15 - addr8 |
|----------------|

|               |
|---------------|
| addr7 - addr0 |
|---------------|

**Operation:** LCALL  
 $(PC) \leftarrow (PC) + 3$   
 $(SP) \leftarrow (SP) + 1$   
 $((SP)) \leftarrow (PC7-0)$   
 $(SP) \leftarrow (SP) + 1$   
 $((SP)) \leftarrow (PC15-8)$   
 $(PC) \leftarrow \text{addr15-0}$

**LJMP addr16**

**Function:** Long Jump

**Description:** LJMP causes an unconditional branch to the indicated address, by loading the high-order and low-order bytes of the PC (respectively) with the second and third instruction bytes. The destination may therefore be anywhere in the full 64K program memory address space. No flags are affected.

**Example:** The label "JMPADR" is assigned to the instruction at program memory location 1234H. The instruction,

**LJMP JMPADR**

at location 0123H will load the program counter with 1234H.

**Bytes:** 3

**Cycles:** 2

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
|---|---|---|---|---|---|---|---|

|                |
|----------------|
| addr15 - addr8 |
|----------------|

|               |
|---------------|
| addr7 - addr0 |
|---------------|

**Operation:** LJMP  
 $(PC) \leftarrow \text{addr15-0}$



## MCS-51 INSTRUCTION SET

**MOV** <dest-byte>, <src-byte> Example: Initially the stack pointer equals 07H. The value of RAM location 07H is 04H. After executing the instruction MOV SP, 07H, the stack pointer will contain 07H and 01H will contain 04H and 02H will contain 03H.

**Function:** Move byte variable

**Description:** The byte variable indicated by the second operand is copied into the location specified by the first operand. The source byte is not affected. No other register or flag is affected.

This is by far the most flexible operation. Fifteen combinations of source and destination addressing modes are allowed.

**Example:** Internal RAM location 30H holds 40H. The value of RAM location 40H is 10H. The data present at input port 1 is 11001010B (0CAH).

```
MOV R0, #30H      ;R0 <= 30H
MOV A, @R0         ;A <= 40H
MOV R1, A          ;R1 <= 40H
MOV R, @R1         ;B <= 10H
MOV @R1, P1        ;RAM (40H) <= 0CAH
MOV P2, P1         ;P2 #0CAH
```

leaves the value 30H in register 0, 40H in both the accumulator and register 1, 10H in register B, and 0CAH (11001010B) both in RAM location 40H and output on port 2.

**MOV A, Rn**

**Bytes:** 1

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 1 | r | r | r |
|---|---|---|---|---|---|---|---|

**Operation:**

MOV  
(A) ← (Rn)

**MOV A, direct**

**Bytes:** 2

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|

direct address

**Operation:**

MOV  
(A) ← (direct)

## MCS-51 INSTRUCTION SET

**MOV A,@Ri**

**Bytes:** 1

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 | 1 | 1 | i |
|---|---|---|---|---|---|---|---|

**Operation:** MOV  
(A) ← ((Ri))

**MOV A,#data**

**Bytes:** 2

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |
|---|---|---|---|---|---|---|---|

**immediate data**

**Operation:** MOV  
(A) ← #data

**MOV Rn,A**

**Bytes:** 1

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | r | r | r |
|---|---|---|---|---|---|---|---|

**Operation:** MOV  
(Rn) ← (A)

**MOV Rn,direct**

**Bytes:** 2

**Cycles:** 2

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 1 | r | r | r |
|---|---|---|---|---|---|---|---|

**direct addr.**

**Operation:** MOV  
(Rn) ← (direct)

**MOV Rn,#data**

**Bytes:** 2

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 | r | r | r |
|---|---|---|---|---|---|---|---|

**immediate data**

**Operation:** MOV  
(Rn) ← #data



## MCS-51 INSTRUCTION SET

### MOV direct,A

Bytes: 2  
Cycles: 1

Encoding:

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|

direct address

Operation:

MOV  
(direct) ← (A)

### MOV direct,Rn

Bytes: 2  
Cycles: 2

Encoding:

|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 1 | r | r | r | r |
|---|---|---|---|---|---|---|---|---|

direct address

Operation:

MOV  
(direct) ← (Rn)

### MOV direct,direct

Bytes: 3  
Cycles: 2

Encoding:

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|

dir. addr. (src)

dir. addr. (dest)

Operation:

MOV  
(direct) ← (direct)

### MOV direct,@Ri

Bytes: 2  
Cycles: 2

Encoding:

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 1 | 1 | i |
|---|---|---|---|---|---|---|---|

direct addr.

Operation:

MOV  
(direct) ← ((Ri))

### MOV direct,#data

Bytes: 3  
Cycles: 2

Encoding:

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|

direct address

immediate data

Operation:

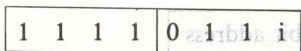
MOV  
(direct) ← #data

## MCS-51 INSTRUCTION SET

### MOV @Ri,A

Bytes: 1  
Cycles: 1

Encoding:



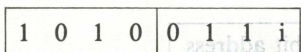
Operation:

MOV  
((Ri)) ← (A)

### MOV @Ri,direct

Bytes: 2  
Cycles: 2

Encoding:



direct addr.

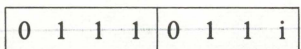
Operation:

MOV  
((Ri)) ← (direct)

### MOV @Ri,#data

Bytes: 2  
Cycles: 1

Encoding:



immediate data

Operation:

MOV  
((Ri)) ← #data

### MOV <dest-bit>,<src-bit>

**Function:** Move bit data

**Description:** The Boolean variable indicated by the second operand is copied into the location specified by the first operand. One of the operands must be the carry flag; the other may be any directly addressable bit. No other register or flag is affected.

**Example:** The carry flag is originally set. The data present at input port 3 is 11000101B. The data previously written to output port 1 is 35H (00110101B).

MOV P1.3,C

MOV C,P3.3

MOV P1.2,C

will leave the carry cleared and change port 1 to 39H (00111001B).



## MCS-51 INSTRUCTION SET

**MOV C,bit**

**Bytes:** 2

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
|---|---|---|---|---|---|---|---|

**bit address**

**Operation:**

MOV  
(C) ← (bit)

**MOV bit,C**

**Bytes:** 2

**Cycles:** 2

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
|---|---|---|---|---|---|---|---|

**bit address**

**Operation:**

MOV  
(bit) ← (C)

**MOV DPTR,#data16**

**Function:**

Load Data Pointer with a 16-bit constant

**Description:**

The data pointer is loaded with the 16-bit constant indicated. The 16-bit constant is loaded into the second and third bytes of the instruction. The second byte (DPH) is the high-order byte, while the third byte (DPL) holds the low-order byte. No flags are affected.

**Example:**

This is the only instruction which moves 16 bits of data at once.  
The instruction,

MOV DPTR,#1234H

will load the value 1234H into the data pointer: DPH will hold 12H and DPL will hold 34H.

**Bytes:** 3

**Cycles:** 2

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|

**immed. data15 - 8**

**immed. data7 - 0**

**Operation:**

MOV  
(DPTR) ← #data15-0  
DPH □ DPL ← #data15-8 □ #data7-0

## MCS-51 INSTRUCTION SET

**MOVC     A,@A + <base-reg>**

- Function:** Move Code byte
- Description:** The MOVC instructions load the accumulator with a code byte, or constant from program memory. The address of the byte fetched is the sum of the original unsigned eight-bit accumulator contents and the contents of a sixteen-bit base register, which may be either the data pointer or the PC. In the latter case, the PC is incremented to the address of the following instruction before being added with the accumulator: otherwise the base register is not altered. Sixteen-bit addition is performed so a carry-out from the low-order eight bits may propagate through higher-order bits. No flags are affected.
- Example:** A value between 0 and 3 is in the accumulator. The following instructions will translate the value in the accumulator to one of four values defined by the DB (define byte) directive.

```
REL_PC: INC      A
          MOVC    A,@A+PC
          RET
          DB      66H
          DB      77H
          DB      88H
          DB      99H
```

If the subroutine is called with the accumulator equal to 01H, it will return with 77H in the accumulator. The INC A before the MOVC instruction is needed to "get around" the RET instruction above the table. If several bytes of code separated the MOVC from the table, the corresponding number would be added to the accumulator instead.

**MOVC     A,@A + DPTR**

**Bytes:** 1

**Cycles:** 2

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|

**Operation:** MOVC  
(A) ← ((A) + (DPTR))



## MCS-51 INSTRUCTION SET

**MOVC A,@A+PC**

**Bytes:** 1

**Cycles:** 2

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|

**Operation:** MOVC  
 $(PC) \leftarrow (PC) + 1$   
 $(A) \leftarrow ((A) + (PC))$

**MOVX <dest-byte>,<src-byte>**

**Function:** Move External

**Description:** The MOVX instructions transfer data between the accumulator and a byte of external data memory, hence the "X" appended to MOV. There are two types of instructions, differing in whether they provide an eight-bit or sixteen-bit indirect address to the external data RAM.

In the first type, the contents of R0 or R1 in the current register bank provide an eight-bit address multiplexed with data on P0. Eight bits are sufficient for external I/O expansion decoding or a relatively small RAM array. For somewhat larger arrays, any output port pins can be used to output higher-order address bits. These pins would be controlled by an output instruction preceding the MOVX.

In the second type of MOVX instruction, the data pointer generates a sixteen-bit address. P2 outputs the high-order eight address bits (the contents of DPH) while P0 multiplexes the low-order eight bits (DPL) with data. The P2 Special Function Register retains its previous contents while the P2 output buffers are emitting the contents of DPH. This form is faster and more efficient when accessing very large data arrays (up to 64K bytes), since no additional instructions are needed to set up the output ports.

It is possible in some situations to mix the two MOVX types. A large RAM array with its high-order address lines driven by P2 can be addressed via the data pointer, or with code to output high-order address bits to P2 followed by a MOVX instruction using R0 or R1.

## MCS-51 INSTRUCTION SET

**Example:** An external 256 byte RAM using multiplexed address/data lines (e.g., an Intel® 8155 RAM/I/O/Timer) is connected to the 8051 Port 0. Port 3 provides control lines for the external RAM. Ports 1 and 2 are used for normal I/O. Registers 0 and 1 contain 12H and 34H. Location 34H of the external RAM holds the value 56H. The instruction sequence,

```
MOVX A,@R1
MOVX @R0,A
```

copies the value 56H into both the accumulator and external RAM location 12H.

**MOVX A,@Ri**

Bytes: 1  
Cycles: 2

Encoding: 

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 | 0 | 1 | i |
|---|---|---|---|---|---|---|---|

Operation: MOVX  
(A) ← ((Ri))

**MOVX A,@DPTR**

Bytes: 1  
Cycles: 2

Encoding: 

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|

Operation: MOVX  
(A) ← ((DPTR))

**MOVX @Ri,A**

Bytes: 1  
Cycles: 2

Encoding: 

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 | 0 | 1 | i |
|---|---|---|---|---|---|---|---|

Operation: MOVX  
((Ri)) ← (A)

**MOVX @DPTR,A**

Bytes: 1  
Cycles: 2

Encoding: 

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|

Operation: MOVX  
(DPTR) ← (A)



## MCS-51 INSTRUCTION SET

### MUL AB

**Function:** Multiply

**Description:** MUL AB multiplies the unsigned eight-bit integers in the accumulator and register B. The low-order byte of the sixteen-bit product is left in the accumulator, and the high-order byte in B. If the product is greater than 255 (0FFH) the overflow flag is set; otherwise it is cleared. The carry flag is always cleared.

**Example:** Originally the accumulator holds the value 80 (50H). Register B holds the value 160 (0A0H). The instruction,

MUL AB

will give the product 12,800 (3200H), so B is changed to 32H (00110010B) and the accumulator is cleared. The overflow flag is set, carry is cleared.

**Bytes:**

1

**Cycles:**

4

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
|---|---|---|---|---|---|---|---|

**Operation:**

MUL

(A)7-0 ← (A) X (B)

(B)15-8

## MCS-51 INSTRUCTION SET

### NOP

**Function:** No Operation

**Description:** Execution continues at the following instruction. Other than the PC, no registers or flags are affected.

**Example:** It is desired to produce a low-going output pulse on bit 7 of port 2 lasting exactly 5 cycles. A simple SETB/CLR sequence would generate a one-cycle pulse, so four additional cycles must be inserted. This may be done (assuming no interrupts are enabled) with the instruction sequence,

CLR P2.7

NOP

NOP

NOP

NOP

SETB P2.7

**Bytes:** 1

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|

**Operation:**

NOP

(PC) ← (PC) + 1



## MCS-51 INSTRUCTION SET

**ORL**     <dest-byte> <src-byte>

**Function:** Logical-OR for byte variables

**Description:** ORL performs the bitwise logical-OR operation between the indicated variables, storing the results in the destination byte. No flags are affected.

The two operands allow six addressing mode combinations. When the destination is the accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the accumulator or immediate data.

*Note:* When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

**Example:** If the accumulator holds 0C3H (11000011B) and R0 holds 55H (01010101B) then the instruction,

ORL     A,R0

will leave the accumulator holding the value 0D7H (11010111B).

When the destination is a directly addressed byte, the instruction can set combinations of bits in any RAM location or hardware register. The pattern of bits to be set is determined by a mask byte, which may be either a constant data value in the instruction or a variable computed in the accumulator at run-time. The instruction,

ORL     P1,#00110010B

will set bits 5, 4, and 1 of output port 1.

**ORL**     A,Rn

**Bytes:**     1

**Cycles:**    1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 1 | r | r | r |
|---|---|---|---|---|---|---|---|

**Operation:**

ORL

$(A) \leftarrow (A) \vee (Rn)$

## MCS-51 INSTRUCTION SET

### ORL A,direct

Bytes: 2

Cycles: 1

Encoding:

|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|---|

direct address

Operation:

ORL

$(A) \leftarrow (A) \vee (\text{direct})$

### ORL A,@Ri

Bytes: 1

Cycles: 1

Encoding:

|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | i |
|---|---|---|---|---|---|---|---|---|

Operation:

ORL

$(A) \leftarrow (A) \vee ((Ri))$

### ORL A,#data

Bytes: 2

Cycles: 1

Encoding:

|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|

immediate data

Operation:

ORL

$(A) \leftarrow (A) \vee \#data$

### ORL direct,A

Bytes: 2

Cycles: 1

Encoding:

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
|---|---|---|---|---|---|---|---|

direct address

Operation:

ORL

$(\text{direct}) \leftarrow (\text{direct}) \vee (A)$

### ORL direct,#data

Bytes: 3

Cycles: 2

Encoding:

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|

direct addr.

immediate data

Operation:

ORL

$(\text{direct}) \leftarrow (\text{direct}) \vee \#data$



## MCS-51 INSTRUCTION SET

### ORL C, <src-bit>

|                    |  |                                  |  |
|--------------------|--|----------------------------------|--|
| <b>Function:</b>   | Logical-OR for bit variables   |                                  |  |
| <b>Description</b> | Set the carry flag if the Boolean value is a logical 1; leave the carry in its current state otherwise. A slash ("/") preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, but the source bit itself is not affected. No other flags are affected. |                                  |  |
| <b>Example:</b>    | Set the carry flag if and only if P1.0 = 1, ACC.7 = 1, or OV = 0:  |                                  |  |
|                    | MOV C,P1.0   | ;LOAD CARRY WITH INPUT PIN P10   |  |
|                    | ORL C,ACC.7  | ;OR CARRY WITH THE ACC. BIT 7    |  |
|                    | ORL C,/OV  | ;OR CARRY WITH THE INVERSE OF OV |  |

**ORL C,bit**  
**Bytes:** 2  
**Cycles:** 2

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |
|---|---|---|---|---|---|---|---|

bit address

**Operation:**

ORL  
 $(C) \leftarrow (C) \vee (\text{bit})$

**ORL C,/bit**  
**Bytes:** 2  
**Cycles:** 2

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|

bit address

**Operation:**

ORL  
 $(C) \leftarrow (C) \vee (\overline{\text{bit}})$

### POP direct

|                     |  |  |  |
|---------------------|--|--|--|
| <b>Function:</b>    | Pop from stack.  |  |  |
| <b>Description:</b> | The contents of the internal RAM location addressed by the stack pointer is read, and the stack pointer is decremented by one. The value read is the transfer to the directly addressed byte indicated. No flags are affected. |  |  |

## MCS-51 INSTRUCTION SET

**Example:** The stack pointer originally contains the value 32H, and internal RAM locations 30H through 32H contain the values 20H, 23H, and 01H, respectively. The instruction sequence,

POP DPH  
POP DPL

will leave the stack pointer equal to the value 30H and the data pointer set to 0123H. At this point the instruction,

POP SP

will leave the stack pointer set to 20H. Note that in this special case the stack pointer was decremented to 2FH before being loaded with the value popped (20H).

**Bytes:** 2  
**Cycles:** 2

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|

|                |
|----------------|
| direct address |
|----------------|

**Operation:** POP  
(direct) ← ((SP))  
(SP) ← (SP) - 1

**PUSH      direct**

**Function:** Push onto stack  
**Description:** The stack pointer is incremented by one. The contents of the indicated variable is then copied into the internal RAM location addressed by the stack pointer. Otherwise no flags are affected.

**Example:** On entering an interrupt routine the stack pointer contains 09H. The data pointer holds the value 0123H. The instruction sequence,

PUSH DPL  
PUSH DPH

will leave the stack pointer set to 0BH and store 23H and 01H in internal RAM locations 0AH and 0BH, respectively.

**Bytes:** 2  
**Cycles:** 2

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|

|                |
|----------------|
| direct address |
|----------------|

**Operation:** PUSH  
(SP) ← (SP) + 1  
((SP)) ← (direct)



## MCS-51 INSTRUCTION SET

|                     |  |   |   |   |   |   |   |   |   |
|---------------------|--|---|---|---|---|---|---|---|---|
| <b>RET</b>          |  |   |   |   |   |   |   |   |   |
| <b>Function:</b>    | Return from subroutine   |   |   |   |   |   |   |   |   |
| <b>Description:</b> | RET pops the high- and low-order bytes of the PC successively from the stack, decrementing the stack pointer by two. Program execution continues at the resulting address, generally the instruction immediately following an ACALL or LCALL. No flags are affected. |   |   |   |   |   |   |   |   |
| <b>Example:</b>     | The stack pointer originally contains the value 0BH. Internal RAM locations 0AH and 0BH contain the values 23H and 01H, respectively. The instruction, RET will leave the stack pointer equal to the value 09H. Program execution will continue at location 0123H.   |   |   |   |   |   |   |   |   |
| <b>Bytes:</b>       | 1  |   |   |   |   |   |   |   |   |
| <b>Cycles:</b>      | 2  |   |   |   |   |   |   |   |   |
| <b>Encoding:</b>    | <table border="1"><tr><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr></table>  | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0                   | 0  | 1 | 0 | 0 | 0 | 1 | 0 |   |   |
| <b>Operation:</b>   | RET<br>$(PC_{15-8}) \leftarrow ((SP))$<br>$(SP) \leftarrow (SP) - 1$<br>$(PC_{7-0}) \leftarrow ((SP))$<br>$(SP) \leftarrow (SP) - 1$   |   |   |   |   |   |   |   |   |

|                     |   |
|---------------------|---|
| <b>RETI</b>         |   |
| <b>Function:</b>    | Return from interrupt   |
| <b>Description:</b> | RETI pops the high- and low-order bytes of the PC successively from the stack, and restores the interrupt logic to accept additional interrupts at the same priority level as the one just processed. The stack pointer is left decremented by two. No other registers are affected; the PSW is <i>not</i> automatically restored to its pre-interrupt status. Program execution continues at the resulting address, which is generally the instruction immediately after the point at which the interrupt request was detected. If a lower- or same-level interrupt had been pending when the RETI instruction is executed, that one instruction will be executed before the pending interrupt is processed. |

## MCS-51 INSTRUCTION SET

**Example:** The stack pointer originally contains the value 0BH. An interrupt was detected during the instruction ending at location 0122H. Internal RAM locations 0AH and 0BH contain the values 23H and 01H, respectively. The instruction, RETI

will leave the stack pointer equal to 09H and return program execution to location 0123H.

**Bytes:** 1

**Cycles:** 2

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
|---|---|---|---|---|---|---|---|

**Operation:**

RETI  
 $(PC_{15-8}) \leftarrow ((SP))$   
 $(SP) \leftarrow (SP) - 1$   
 $(PC_{7-0}) \leftarrow ((SP))$   
 $(SP) \leftarrow (SP) - 1$

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 |
|---|---|---|---|---|---|---|---|

**Function:** Rotate accumulator left through the Carry flag.  
**Description:** The eight bits in the accumulator and the carry flag are together rotated one bit to the left. Bit 7 moves into the carry flag; the original state of the carry flag moves into the bit 0 position. No other flags are affected.  
**Example:** The accumulator holds the value 0C3H (10001011B), and the carry is zero. The instruction, RLC A

leaves the accumulator holding the value 8BH (10001010B) with the carry set.

**Bytes:** 1  
**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|

**Operation:**

RLC  
 $(A_{n+1}) \leftarrow (A_n)$   
 $n = 0 - 6$   
 $(A_0) \leftarrow (C)$   
 $(C) \leftarrow (A_7)$



## MCS-51 INSTRUCTION SET

### RL     A

**Function:** Rotate accumulator Left

**Description:** The eight bits in the accumulator are rotated one bit to the left. Bit 7 is rotated into the bit 0 position. No flags are affected.

**Example:** The accumulator holds the value 0C5H (11000101B). The instruction,

RL     A

leaves the accumulator holding the value 8BH (10001011B) with the carry unaffected.

**Bytes:**

1

**Cycles:**

1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|

**Operation:**

RL

$(A_{n+1}) \leftarrow (A_n) \quad n = 0-6$

$(A_0) \leftarrow (A_7)$

### RLC     A

**Function:** Rotate accumulator Left through the Carry flag

**Description:** The eight bits in the accumulator and the carry flag are together rotated one bit to the left. Bit 7 moves into the carry flag; the original state of the carry flag moves into the bit 0 position. No other flags are affected.

**Example:** The accumulator holds the value 0C5H (11000101B), and the carry is zero. The instruction,

RLC     A

leaves the accumulator holding the value 8BH (10001010B) with the carry set.

**Bytes:**

1

**Cycles:**

1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|

**Operation:**

RLC

$(A_{n+1}) \leftarrow (A_n) \quad n = 0-6$

$(A_0) \leftarrow (C)$

$(C) \leftarrow (A_7)$

## MCS-51 INSTRUCTION SET

**RR     A**

**Function:** Rotate accumulator Right

**Description:** The eight bits in the accumulator are rotated one bit to the right. Bit 0 is rotated into the bit 7 position. No flags are affected.

**Example:** The accumulator holds the value 0C5H (11000101B). The instruction,

RR     A

leaves the accumulator holding the value 0E2H (11100010B) with the carry unaffected.

**Bytes:** 1

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|

**Operation:**

RR

$(A_n) \leftarrow (A_n + 1) \quad n = 0 - 6$

$(A_7) \leftarrow (A_0)$

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|

**RRC     A**

**Function:** Rotate accumulator Right through Carry flag

**Description:** The eight bits in the accumulator and the carry flag are together rotated one bit to the right. Bit 0 moves into the carry flag; the original value of the carry flag moves into the bit 7 position. No other flags are affected.

**Example:** The accumulator holds the value 0C5H (11000101B), the carry is zero. The instruction,

RRC     A

leaves the accumulator holding the value 62 (01100010B) with the carry set.

**Bytes:** 1

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|

**Operation:**

RRC

$(A_n) \leftarrow (A_n + 1) \quad n = 0 - 6$

$(A_7) \leftarrow (C)$

$(C) \leftarrow (A_0)$



## MCS-51 INSTRUCTION SET

### SETB <bit>

**Function:** Set Bit

**Description:** SETB sets the indicated bit to one. SETB can operate on the carry flag or any directly addressable bit. No other flags are affected.

**Example:** The carry flag is cleared. Output port 1 has been written with the value 34H (00110100B). The instructions,

```
SETB C
SETB P1.0
```

will leave the carry flag set to 1 and change the data output on port 1 to 35H (00110101B).

#### SETB C

**Bytes:** 1

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|

**Operation:**

SETB  
(C) ← 1

#### SETB bit

**Bytes:** 2

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
|---|---|---|---|---|---|---|---|

bit address

**Operation:**

SETB  
(bit) ← 1

### SJMP rel

**Function:** Short Jump

**Description:** Program control branches unconditionally to the address indicated. The branch destination is computed by adding the signed displacement in the second instruction byte to the PC, after incrementing the PC twice. Therefore, the range of destinations allowed is from 128 bytes preceding this instruction to 127 bytes following it.

## MCS-51 INSTRUCTION SET

**Example:** The label "RELADR" is assigned to an instruction at program memory location 0123H. The instruction,

SJMP RELADR

will assemble into location 0100H. After the instruction is executed, the PC will contain the value 0123H.

(Note: Under the above conditions the instruction following SJMP will be at 102H. Therefore, the displacement byte of the instruction will be the relative offset  $(0123H - 0102H) = 21H$ . Put another way, an SJMP with a displacement of 0FEH would be a one-instruction infinite loop.)

**Bytes:** 2

**Cycles:** 2

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|

rel. address

**Operation:**

SJMP

$(PC) \leftarrow (PC) + 2$

$(PC) \leftarrow (PC) + \text{rel}$



## MCS-51 INSTRUCTION SET

### SUBB    A, <src-byte>

**Function:** Subtract with borrow

**Description:** SUBB subtracts the indicated variable and the carry flag together from the accumulator, leaving the result in the accumulator. SUBB sets the carry (borrow) flag if a borrow is needed for bit 7, and clears C otherwise. (If C was set *before* executing a SUBB instruction, this indicates that a borrow was needed for the previous step in a multiple precision subtraction, so the carry is subtracted from the accumulator along with the source operand.) AC is set if a borrow is needed for bit 3, and cleared otherwise. OV is set if a borrow is needed into bit 6, but not into bit 7, or into bit 7, but not bit 6.

When subtracting signed integers OV indicates a negative number produced when a negative value is subtracted from a positive value, or a positive result when a positive number is subtracted from a negative number.

The source operand allows four addressing modes: register, direct, register-indirect, or immediate.

**Example:** The accumulator holds 0C9H (11001001B), register 2 holds 54H (01010100B), and the carry flag is set. The instruction,

SUBB    A,R2

will leave the value 74H (01110100B) in the accumulator, with the carry flag and AC cleared but OV set.

Notice that 0C9H minus 54H is 75H. The difference between this and the above result is due to the carry (borrow) flag being set before the operation. If the state of the carry is not known before starting a single or multiple-precision subtraction, it should be explicitly cleared by a CLR C instruction.

**SUBB    A,Rn**

**Bytes:** 1

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 1 | r | r | r |
|---|---|---|---|---|---|---|---|

**Operation:**

SUBB

$(A) \leftarrow (A) - (C) - (Rn)$

## MCS-51 INSTRUCTION SET

### SUBB A, direct

Bytes: 2

Cycles: 1

Encoding: 

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|

 direct address

Operation: SUBB  
(A) ← (A) - (C) - (direct)

### SUBB A, @Ri

Bytes: 1

Cycles: 1

Encoding: 

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 0 | 1 | 1 | i |
|---|---|---|---|---|---|---|---|

Operation: SUBB  
(A) ← (A) - (C) - ((Ri))

### SUBB A, #data

Bytes: 2

Cycles: 1

Encoding: 

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
|---|---|---|---|---|---|---|---|

 immediate data

Operation: SUBB  
(A) ← (A) - (C) - #data

### SWAP A

Function: Swap nibbles within the Accumulator

Description: SWAP A interchanges the low- and high-order nibbles (four-bit fields) of the accumulator (bits 3-0 and bits 7-4). The operation can also be thought of as a four-bit rotate instruction. No flags are affected.

Example: The accumulator holds the value 0C5H (11000101B). The instruction,  
SWAP A

leaves the accumulator holding the value 5CH (01011100B).

Bytes: 1

Cycles: 1

Encoding: 

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
|---|---|---|---|---|---|---|---|

Operation: SWAP  
(A3-0) ↔ (A7-4), (A7-4) ← (A3-0)



## MCS-51 INSTRUCTION SET

### XCH    A,<byte>

**Function:** Exchange Accumulator with byte variable  
**Description:** XCH loads the accumulator with the contents of the indicated variable, at the same time writing the original accumulator contents to the indicated variable. The source/destination operand can use register, direct, or register-indirect addressing.

**Example:** R0 contains the address 20H. The accumulator holds the value 3FH (00111111B). Internal RAM location 20H holds the value 75H (01110101B). The instruction,

XCH    A,@R0

will leave RAM location 20H holding the values 3FH (00111111B) and 75H (01110101B) in the accumulator.

### XCH    A,Rn

**Bytes:** 1

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 1 | r | r | r |
|---|---|---|---|---|---|---|---|

**Operation:**

XCH  
 (A)  $\longleftrightarrow$  (Rn)

### XCH    A,direct

**Bytes:** 2

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |                |
|---|---|---|---|---|---|---|---|----------------|
| 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | direct address |
|---|---|---|---|---|---|---|---|----------------|

**Operation:**

XCH  
 (A)  $\longleftrightarrow$  (direct)

### XCH    A,@Ri

**Bytes:** 1

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 1 | 1 | i |
|---|---|---|---|---|---|---|---|

**Operation:**

XCH  
 (A)  $\longleftrightarrow$  ((Ri))

## MCS-51 INSTRUCTION SET

**XCHD      A,@Ri**

**Function:** Exchange Digit  
**Description:** XCHD exchanges the low-order nibble of the accumulator (bits 3-0), generally representing a hexadecimal or BCD digit), with that of the internal RAM location indirectly addressed by the specified register. The high-order nibbles (bits 7-4) of each register are not affected. No flags are affected.  
**Example:** R0 contains the address 20H. The accumulator holds the value 36H (00110110B). Internal RAM location 20H holds the value 75H (01110101B). The instruction,

XCHD      A,@R0

will leave RAM location 20H holding the value 76H (01110110B) and 35H (00110101B) in the accumulator.

**Bytes:** 1  
**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 0 | 1 | 1 | i |
|---|---|---|---|---|---|---|---|

**Operation:** XCHD  
 $(A3-0) \leftrightarrow ((Ri3-0))$



## MCS-51 INSTRUCTION SET

**XRL**     <dest-byte>, <src-byte>

**Function:** Logical Exclusive-OR for byte variables

**Description:** XRL performs the bitwise logical Exclusive-OR operation between the indicated variables, storing the results in the destination. No flags are affected.

The two operands allow six addressing mode combinations. When the destination is the accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the accumulator or immediate data.

(Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.)

**Example:** If the accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B) then the instruction,

XRL     A,R0

will leave the accumulator holding the value 69H (01101001B).

When the destination is a directly addressed byte, this instruction can complement combinations of bits in any RAM location or hardware register. The pattern of bits to be complemented is then determined by a mask byte, either a constant contained in the instruction or a variable computed in the accumulator at run-time. The instruction,

XRL     P1,#00110001B

will complement bits 5, 4, and 0 of output port 1.

**XRL**     A,Rn

**Bytes:**     1

**Cycles:**    1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 1 | r | r | r |
|---|---|---|---|---|---|---|---|

**Operation:**

XRL

$(A) \leftarrow (A) \vee (Rn)$

**XRL**     A,direct

**Bytes:**     2

**Cycles:**    1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|

direct address

**Operation:**

XRL

$(A) \leftarrow (A) \vee (\text{direct})$

## MCS-51 INSTRUCTION SET

---

**XRL A,@Ri**

**Bytes:** 1

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | i |
|---|---|---|---|---|---|---|---|

**Operation:**

XRL

$(A) \leftarrow (A) \vee ((\dot{R}i))$

**XRL A,#data**

**Bytes:** 2

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
|---|---|---|---|---|---|---|---|

immediate data

**Operation:**

XRL

$(A) \leftarrow (A) \vee \#data$

**XRL direct,A**

**Bytes:** 2

**Cycles:** 1

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |
|---|---|---|---|---|---|---|---|

direct address

**Operation:**

XRL

$(direct) \leftarrow (direct) \vee (A)$

**XRL direct,#data**

**Bytes:** 3

**Cycles:** 2

**Encoding:**

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|

direct address

immediate data

**Operation:**

XRL

$(direct) \leftarrow (direct) \vee \#data$



---

# **MCS<sup>®</sup>-51 Application Examples**

---

**9**

## CHAPTER 9

# MCS<sup>®</sup>-51 APPLICATION EXAMPLES

Chapter 9 contains three sections:

- 8051 Programming Techniques
- Peripheral Interfacing Techniques
- Connecting to Peripherals

The first section has 8051 software examples for some common routines in controller applications. Some routines included are multiple-precision arithmetic and table look-up techniques.

Peripheral Interfacing Techniques include routines for handling the 8051's I/O ports, serial channel and timer/counters. Discussed in this section is I/O port reconfiguration, software delay timing, and transmitting serial

port character strings along with other routines.

## 9.0 8051 PROGRAMMING TECHNIQUES

### 9.0.1 Radix Conversion Routines

The divide instruction can be used to convert a number from one radix to another. BINBCD is a short subroutine to convert an eight-bit unsigned binary integer in the accumulator (between 0 & 255) to a three-digit (two byte) BCD representation. The hundred's digit is returned in one variable (HUND) and the ten's and one's digits returned as packed BCD in another (TENONE).

```

;
;BINBCD  CONVERT 8-BIT BINARY VARIABLE IN ACCUMULATOR
;        TO 3-DIGIT PACKED BCD FORMAT.
;        HUNDREDS' PLACE LEFT IN VARIABLE 'HUND',
;        TENS' AND ONES' PLACES IN 'TENONE'.
;
HUND    DATA    21H
TENONE  DATA    22H
;
BINBCD:  MOV     B,#100          ;DIVIDED BY 100 TO
        DIV     AB              ;DETERMINE NUMBER OF HUNDREDS
        MOV     HUND,A
        MOV     A,#10          ;DIVIDE REMAINDER BY TEN TO
        XCH     A,B            ;DETERMINE NUMBER OF TENS LEFT
        DIV     AB             ;TEN'S DIGIT IN ACC, REMAINDER IS
                                ;ONE'S DIGIT
        SWAP    A
        ADD     A,B            ;PACK BCD DIGITS IN ACC
        MOV     TENONE,A
        RET
;

```



## MCS-51 APPLICATION EXAMPLES

The divide instruction can also separate data in the accumulator into sub-fields. For example, dividing packed BCD data by 16 will separate the two nibbles, leaving the high-order digit in the accumulator and the low-order digit (remainder) in B. Each is right-justified, so the

digits can be processed individually. This example receives two packed BCD digits in the accumulator, separates the digits, computes their product, and returns the product in packed BCD format in the accumulator.

```
;
;MULBCD  UNPACK TWO BCD DIGITS RECEIVED IN ACCUMULATOR,
;        FIND THEIR PRODUCT, AND RETURN PRODUCT
;        IN PACKED BCD FORMAT IN ACCUMULATOR
;
MULBCD: MOV    B,#10H      ;DIVIDE INPUT BY 16
        DIV    AB          ;A & B HOLD SEPARATED DIGITS
                        ;(EACH RIGHT JUSTIFIED IN REGISTER).
        MUL    AB          ;A HOLDS PRODUCT IN BINARY FORMAT (0-
                        ;99 (DECIMAL) = 0 — 63H)
        MOV    B,#10      ;DIVIDE PRODUCT BY 10
        DIV    AB          ;A HOLDS NUMBER OF TENS, B HOLDS
                        ;REMAINDER
        SWAP   A
        ORL    A,B        ;PACK DIGITS
        RET
```

### 9.0.2 Multiple Precision Arithmetic

The ADDC and SUBB instructions incorporate the previous state of the carry (borrow) flag to allow multiple-precision calculations by repeating the operation with successively higher-order operand bytes. If the input data for a multiple-precision operation is an unsigned string of integers, the carry flag will be

set upon completion if an overflow (for ADDC) or underflow (for SUBB) occurs. With two's complement signed data, the most significant bit of the original input data's most significant byte indicates the sign of the string, so the overflow flag (OV) will indicate if overflow or underflow occurred.

```
;
;SUBSTR  SUBTRACT STRING INDICATED BY R1
;        FROM STRING INDICATED BY R0 TO
;        PRECISION INDICATED BY R2.
;        CHECK FOR SIGNED UNDERFLOW WHEN DONE.
;
SUBSTR: CLR    C           ;BORROW = 0.
```

## MCS-51 APPLICATION EXAMPLES

---

```

SUBS1:  MOV  A,@R0      ;LOAD MINUEND BYTE
        SUBB  A,@R1     ;SUBTRACT SUBTRAHEND BYTE
        MOV   @R0,A     ;STORE DIFFERENCE BYTE
        INC   R0        ;BUMP POINTERS TO NEXT PLACE
        INC   R1
        DJNZ  R2,SUBS1   ;LOOP UNTIL DONE
;
;
;      WHEN DONE, TEST IF OVERFLOW OCCURRED
;      ON LAST ITERATION OF LOOP.
;
;
        JNB   OV,OV_OK   ;
;
        ...             ;(OVERFLOW RECOVERY ROUTINE)
        OV-OK: RET      ;RETURN

```

---

### 9.0.3 Table Look-Up Sequences

The two versions of the MOVC instructions are used as part of a three-step sequence to access look-up tables in ROM. To use the DPTR version, load the Data Pointer with the starting address of a look-up table; load the accumulator with (or compute) the index of the entry desired; and execute MOVC A,@A+DPTR. The data pointer may be loaded with a constant for short tables, or to allow more complicated data structures, and tables with more than 256 entries, the values for DPH and DPL may be computed or modified with the standard arithmetic instruction set.

The PC-based version is used with smaller, "local" tables, and has the advantage of not affecting the data pointer. This makes it useful in interrupt routines or other situations where the DPTR contents might be significant. Again, a look-up sequence takes three steps: load the accumulator with the index; compensate for the offset from the look-up instruction's address to the start of the table by adding that offset to the accumulator; then execute the MOVC A,@A+PC instruction.

As a non-trivial situation where this instruction would be used, consider applications which store large multi-dimensional look-up tables of dot matrix patterns, non-linear calibration parameters, and so on in the linear (one-dimensional) program memory. To retrieve data from the tables, variables representing matrix indices must be converted to the desired entry's memory address. For a matrix of dimensions (MDIMEN x NDIMEN) starting at address BASE and respective indices INDEXI and INDEXJ, the address of element (INDEXI, INDEXJ) is determined by the formula,

$$\text{Entry Address} = [\text{BASE} + (\text{NDIMEN} \times \text{INDEXI}) + \text{INDEXJ}]$$

The subroutine MATRX1 can access an entry in any array with less than 255 elements (e.g., an 11x21 array with 231 elements). The table entries are defined using the Data Byte ("DB") directive, and will be contained in the assembly object code as part of the accessing subroutine itself.



## MCS-51 APPLICATION EXAMPLES

To handle the more general case, subroutine MATRIX2 allows tables to be unlimited in size, by combining the MUL instruction, double-precision addition, and the data pointer-based version of MOVC. The only restriction is that each index be between 0 and 255.

```

;
;MATRIX1  LOAD CONSTANT READ FROM TWO DIMENSIONAL LOOK-UP
;          TABLE IN PROGRAM MEMORY INTO ACCUMULATOR
;          USING LOCAL TABLE LOOK-UP INSTRUCTION, 'MOVC A,@A+PC'.
;          THE TOTAL NUMBER OF TABLE ENTRIES IS ASSUMED TO
;          BE SMALL, I.E. LESS THAN ABOUT 255 ENTRIES.
;          TABLE USED IN THIS EXAMPLE IS 11 x 21.
;          DESIRED ENTRY ADDRESS IS GIVEN BY THE FORMULA,
;
;          [(BASE ADDRESS) + (21 X INDEXI) + (INDEXJ)]
;
INDEXI    EQU    R6      ;FIRST COORDINATE OF ENTRY (0-10).
INDEXJ    DATA  23H    ;SECOND COORDINATE OF ENTRY (0-20).
MATRIX1:  MOV     A,INDEXI
          MOV     B,#21
          MUL     AB      ;(21 X INDEXI)
          ADD     A,INDEXJ ;ADD IN OFFSET WITHIN ROW
;
;          ALLOW FOR INSTRUCTION BYTE BETWEEN "MOVC" AND
;          ENTRY (0,0).
          INC     A
          MOVC    A,@A+PC
          RET
BASE1:    DB      1       ;(entry 0,0)
          DB      2       ;(entry 0,1)
          ..
          DB      21      ;(entry 0,20)
          DB      22      ;(entry 1,0)
          ..
          DB      42      ;(entry 1,20)
          ..
          DB      231     ;(entry 10,20)

```

## MCS-51 APPLICATION EXAMPLES

```

MATRX2:  MOV    A,INDEXI      ;LOAD FIRST COORDINATE
         MOV    B,#NDIMEN
         MUL    AB             ;INDEXI X NDIMEN
         ADD    A,#LOW(BASE2)  ;ADD IN 16-BIT BASE ADDRESS
         MOV    DPL,A
         MOV    A,B
         ADDC   A,#HIGH(BASE2)
         MOV    DPH,A          ;DPTR=(BASE ADDR) + (INDEXI + NDIMEN)
         MOV    A,INDEXJ
         MOVC   A,@A + DPTR    ;ADD INDEXJ AND FETCH BYTE
         RET

BASE2:   ...      .....
         DB      0          ;(entry 0,0)
         DB      0          ;(entry 0,1)
;
         ...      .....
         DB      0          ;(entry 0, NDIMEN-1)
         DB      0          ;(entry 1,0)
;
         ...      .....
         DB      0          ;(entry 1, NDIMEN-1)
;
         ...      .....
         DB      0          ;(entry MDIMEN-1, NDIMEN-1)

```

### 9.0.4 Saving CPU Status during Interrupts

When the 8051 hardware recognizes an interrupt request, program control branches automatically to the corresponding service routine, by forcing the CPU to process a Long CALL (LCALL) instruction to the appropriate address. The return address is stored on the top of the stack. After completing the service routine, an RETI instruction returns the processor to the background program at the point from which it was interrupted.

Interrupt service routines must not change any variable or hardware registers modified by the main program, or else the program may not resume correctly. (Such a change might look like a spontaneous random error. An example of this will be given later in this section, in the second method of I/O port reconfiguration.) Resources used or altered by the service routine (Accumulator, PSW, etc.) must be saved and restored to their previous value before returning from the service routine. PUSH and POP provide an efficient and convenient way to save such registers on the stack.





## MCS-51 APPLICATION EXAMPLES

```

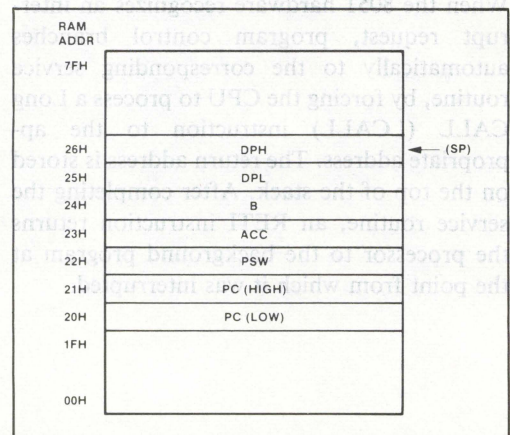
;
LOC_TMPEQU    $           ;REMEMBER LOCATION COUNTER
;
                ORG    0003H           ;STARTING ADDRESS FOR INTERRUPT ROUTINE
                LJMP   SERVER          ;JUMP TO ACTUAL SERVICE ROUTINE LOCATE
                                        ;ELSEWHERE
;
...           .....
                ORG    LOC_TMP        ;RESTORE LOCATION COUNTER
SERVER:        PUSH   PSW             ;SAVE ACCUMULATOR (NOTE DIRECT ADDRESS
                PUSH   ACC            ;NOTATION)
                PUSH   B              ;SAVE B REGISTER
                PUSH   DPL            ;SAVE DATA POINTER
                PUSH   DPH            ;
                MOV    PSW,#00001000B ;SELECT REGISTER BANK 1
;
...           .....
;
                POP    DPH            ;RESTORE REGISTERS IN REVERSE ORDER
                POP    DPL
                POP    B
                POP    ACC
                POP    PSW            ;RESTORE PSW AND RE-SELECT ORIGINAL
                                        ;REGISTER BANK
                RETI                  ;RETURN TO MAIN PROGRAM AND RESTORE
                                        ;INTERRUPT LOGIC

```

If the SP register held 1FH when the interrupt was detected, then while the service routine was in progress the stack would hold the registers shown in Figure 9-1; SP would contain 26H. This is the most general case; if the service routine doesn't alter the B-register and data pointer, for example, the instructions saving and restoring those registers could be omitted.

### 9.0.5 Passing Parameters on the Stack

The stack may also pass parameters to and from subroutines. The subroutine can indirectly address the parameters derived from the contents of the stack pointer, or simply pop the stack into registers before processing.



**Figure 9-1. Stack contents during interrupt**

## MCS-51 APPLICATION EXAMPLES

```

HEXASC: MOV    R0,SP
        DEC    R0                ;ACCESS LOCATION PARAMETER PUSHED INTO
        DEC    R0                ;CONVERT HUNDREDS DIGIT
        XCH    A,@R0            ;READ INPUT PARAMETER AND SAVE AC-
                                ;CUMULATOR
        ANL    A,#0FH           ;MASK ALL BUT LOW-ORDER 4 BITS
        ADD    A,#2             ;ALLOW FOR OFFSET FROM MOV C TO TABLE
        MOVC   A,@A + PC        ;READ LOOK-UP TABLE ENTRY
        XCH    A,@R0            ;PASS BACK TRANSLATED VALUE AND RESTORE
                                ;ACCUMULATOR
        RET                     ;RETURN TO BACKGROUND PROGRAM

ASCTBL: DB      '0'             ;ASCII CODE FOR 00H
        DB      '1'             ;ASCII CODE FOR 01H
        DB      '2'             ;ASCII CODE FOR 02H
        DB      '3'             ;ASCII CODE FOR 03H
        DB      '4'             ;ASCII CODE FOR 04H
        DB      '5'             ;ASCII CODE FOR 05H
        DB      '6'             ;ASCII CODE FOR 06H
        DB      '7'             ;ASCII CODE FOR 07H
        DB      '8'             ;ASCII CODE FOR 08H
        DB      '9'             ;ASCII CODE FOR 09H
        DB      'A'             ;ASCII CODE FOR 0AH
        DB      'B'             ;ASCII CODE FOR 0BH
        DB      'C'             ;ASCII CODE FOR 0CH
        DB      'D'             ;ASCII CODE FOR 0DH
        DB      'E'             ;ASCII CODE FOR 0EH
        DB      'F'             ;ASCII CODE FOR 0FH
    
```

One advantage here is simplicity. Variables need not be allocated for specific parameters, a potentially large number of parameters may be passed, and different calling programs may use different techniques for determining or handling the variables.

For example, the subroutine **HEXASC** converts a hexadecimal value to ASCII code for its low-order digit. It first reads a parameter stored on the stack by the calling program, then uses the low-order bits to access a local 16-entry look-up table holding ASCII codes, stores the appropriate code back in the

stack and then returns. The accumulator contents are left unchanged.

The background program may reach this subroutine with several different calling sequences, all of which **PUSH** a value before calling the routine and **POP** the result to any destination register or port later. There is even the option of leaving a value on the stack if it won't be needed until later. The example below converts the three-digit BCD value computed in the Radix Conversion example above to a three-character string, calling a subroutine **SP\_OUT** to output an eight-bit code in the accumulator.



## MCS-51 APPLICATION EXAMPLES

---

```
;
...      .....
PUSH    HUND      ;CONVERT HUNDREDS DIGIT
CALL    HEXASC
POP     ACC
CALL    SP_OUT    ;TRANSMIT HUNDREDS CHARACTER
PUSH    TENONE
CALL    HEXASC    ;CONVERT ONE'S PLACE DIGIT
                        ;BUT LEAVE ON STACK!
MOV     A, TENONE
SWAP    A          ;RIGHT-JUSTIFY TEN'S PLACE
PUSH    ACC        ;CONVERT TEN'S PLACE DIGIT
CALL    HEXASC
POP     ACC
CALL    SP_OUT    ;TRANSMIT TEN'S PLACE CHARACTER
POP     ACC
CALL    SP_OUT    ;TRANSMIT ONE'S PLACE CHARACTER
...      .....
```

---

### 9.0.6 N-Way Branching

There are several different means for branching to sections of code determined or selected at run time. (The single destination addresses incorporated into conditional and unconditional jumps are, of course, fixed at assembly time.) Each has advantages for different applications.

In a typical N-way branch situation, the potential destinations are generally known at assembly time. One of a number of small routines is selected according to the value of an index variable determined while the program is running. The most efficient way to solve this problem is with the MOVC and an indirect jump instruction, using a short table of offset values in ROM to indicate the relative starting addresses of the several routines.

JMP @A+DPTR is an instruction which performs an indirect jump to an address determined during program execution. The

instruction adds the eight-bit unsigned accumulator contents with the contents of the sixteen-bit data pointer, just like MOVC A,@A+DPTR. The resulting sum is loaded into the program counter and is used as the address for subsequent instruction fetches. Again, a sixteen-bit addition is performed: a carry-out from the low-order eight-bits may propagate through the higher-order bits. In this case, neither the accumulator contents nor the data pointer is altered.

The example subroutine below reads a byte of RAM into the accumulator from one of four alternate address spaces, as selected by the contents of the variable MEMSEL. The address of the byte to be read is determined by the contents of R0 (and optionally R1). It might find use in a printing terminal application, where four different model printers all use the same ROM code but use different types (and sizes) of buffer memory for different speeds and options.

## MCS-51 APPLICATION EXAMPLES

```

;
MEMSEL EQU R3
;
JUMP_4: MOV A, MEMSEL
        MOV DPTR, #JMPTBL
        MOVC A, @A + DPTR
        JMP @A + DPTR
JMPTBL: DB MEMSP0-JMPTBL
        DB MEMSP1-JMPTBL
        DB MEMSP2-JMPTBL
        DB MEMSP3-JMPTBL
MEMSP0: MOV A, @R0 ;READ FROM INTERNAL RAM
        RET
MEMSP1: MOVX A, @R0 ;READ 256 BYTE EXTERNAL RAM
        RET
MEMSP2: MOV DPL, R0 ;READ 64K BYTE EXTERNAL RAM
        MOV DPH, R1
        MOVX A, @DPTR
        RET
MEMSP3: MOV A, R1 ;READ 4K BYTE EXTERNAL RAM
        ANL A, #07H
        ANL P1, #11111000B
        ORL P1, A
        MOVX A, @R0
        RET

```

To use this approach, the size of the jump table plus the length of the alternate routines must be less than 256 bytes. The jump table and routines may be located anywhere in program memory and are independent of 256-byte program memory pages.

For applications where up to 128 destinations must be selected, all residing in the same 2K page of program memory, the following technique may be used. In the printing terminal example, this sequence could process 128 different codes for ASCII characters arriving via the 8051 serial port.



## MCS-51 APPLICATION EXAMPLES

```

;
OPTION EQU R3
;
;
;
JMP128: MOV A,OPTION
        RL A ;MULTIPLY BY 2 FOR 2-BYTE JUMP TABLE
        MOV DPTR,#INSTBL ;FIRST ENTRY IN JUMP TABLE
        JMP @A+DPTR ;JUMP INTO JUMP TABLE
;
;
INSTBL: AJMP PROC00 ;128 CONSECUTIVE
        AJMP PROC01 ;AJMP INSTRUCTIONS
        AJMP PROC02
;
;
;
        AJMP PROC7E
        AJMP PROC7F

```

The destinations in the jump table (PROC00-PROC7F) are not all necessarily unique routines. A large number of special control codes could each be processed with their own unique routine, with the remaining printing characters all causing a branch to a common routine for entering the character into the output queue.

### 9.0.7 Computing Branch Destinations at Run Time

In some rare situations, 128 options are insufficient, the destination routines may cross a 2K page boundary, or a branch destination is not known at assembly time (for whatever reason), and therefore cannot be easily included in the assembled code. These situations can all be

handled by computing the destination address at run-time with standard arithmetic or table look-up instructions, then performing an indirect branch to that address. There are two simple ways to execute this last step, assuming the 16-bit destination address has already been computed. The first is to load the address into the DPH and DPL registers, clear the accumulator and branch using the JMP @A+DPTR instruction; the second is to push the destination address onto the stack, low-order byte first (so as to mimic a call instruction) then pop that address into the PC by performing a return instruction. This also adjusts the stack pointer to its previous value. The code segment below illustrates the latter possibility.

```

;
RTEMP EQU R7
;
;
JMP256: MOV DPTR,#ADRTBL ;FIRST ADDRESS TABLE ENTRY
        MOV A,OPTION ;LOAD INDEX INTO TABLE
        CLR C
        RLC A ;MULTIPLY BY 2 FOR 2-BYTE JUMP TABLE
        JNC LOW128
        INC DPH ;FIX BASE IF INDEX >127.

```

## MCS-51 APPLICATION EXAMPLES

```

LOW128:  MOV    RTEMP,A          ;SAVE ADJUSTED ACC FOR SECOND READ
         INC     A                ;READ LOW-ORDER BYTE FIRST
         MOVC   A,@A+DPTR        ;GET LOW-ORDER BYTE FROM TABLE
         PUSH   ACC              ;
         MOV    A,RTEMP          ;RELOAD ADJUSTED ACC
         MOVC   A,@A+DPTR        ;GET HIGH-ORDERED BYTE FROM TABLE
         PUSH   ACC              ;
;
;      THE TWO ACC PUSHES HAVE PRODUCED
;      A "RETURN ADDRESS" ON THE STACK WHICH CORRESPONDS
;      TO THE DESIRED STARTING ADDRESS.
;      IT MAY BE REACHED BY POPPING THE STACK
;      INTO THE PC.
         RET
;
;      ...
;      ...
;      ...
ADRTBL:  DW     PROC00           ;UP TO 256 CONSECUTIVE DATA
         DW     PROC01           ;WORDS INDICATING STARTING ADDRESSES
;
;      ...
;      ...
         DW     PROCFF
;

```

### 9.0.8 In-Line-Code Parameter Passing

Parameters can be passed by loading appropriate registers with values before calling the subroutine. This technique is inefficient if a lot of the parameters are constants, since each would require a separate register to carry it, and a separate instruction to load the register each time the routine is called.

If the routine is called frequently, a more code-efficient way to transfer constants is "in-line-code" parameter-passing. The constants are actually part of the program code, immediately following the call instruction. The subroutine determines where to find them from the return address on the stack, and then reads the parameters it needs from program memory.

For example, assume a utility named ADD-

BCD adds a 16-bit packed-BCD constant with a two-byte BCD variable in internal RAM and stores the sum in a different two-byte buffer. The utility must be given the constant and both buffer addresses. Rather than using four working registers to carry this information, all four bytes could be inserted into program memory each time the utility is called. Specifically, the calling sequence below invokes the utility to add 1234 (decimal) with the string at internal RAM address 56H, and store the sum in a buffer at location 78H.

The ADDBCD subroutine determines at what point the call was made by popping the return address from the stack into the data pointer high- and low-order bytes. A MOVC instruction then reads the parameters from program memory as they are needed. When done, ADDBCD resumes execution by jumping to the instruction following the last parameter.



## MCS-51 APPLICATION EXAMPLES

```

...      .....
CALL     ADDBCD
DW       1234H      ;BCD CONSTANT
DB       56H        ;SOURCE STRING ADDRESS
DB       78H        ;DESTINATION STRING ADDRESS
...      .....      ;CONTINUATION OF PROGRAM

;
;
;
ADDBCD:  POP     DPH      ;POP RETURN ADDRESS INTO DPTR
        POP     DPL
        MOV     A,#2     ;INDEX FOR SOURCE STRING PARAMETER
        MOVC   A,@A+DPTR ;GET SOURCE STRING LOCATION
        MOV     R0,A
        MOV     A,#3     ;INDEX FOR DESTINATION STRING PARAMETER
        MOVC   A,@A+DPTR ;GET DESTINATION ADDRESS
        MOV     R1,A
        MOV     A,#1     ;INDEX FOR 16-BIT CONSTANT LOW BYTE
        MOVC   A,@A+DPTR ;GET LOW-ORDER VALUE
        ADD     A,@R0     ;COMPUTE LOW-ORDER BYTE OF SUM
        DA      A        ;DECIMAL ADJUST FOR ADDITION
        MOV     @R1,A    ;SAVE IN BUFFER
        INC     R0
        INC     R1
        CLR     A        ;INDEX FOR HIGH-BYTE = 0
        MOVC   A,@A+DPTR ;GET HIGH-ORDER CONSTANT
        ADDC    A,@R0
        DA      A        ;DECIMAL ADJUST FOR ADDITION
        MOV     @R1,A    ;SAVE IN BUFFER
        MOV     A,#4     ;INDEX FOR CONTINUATION OF PROGRAM
        JMP     @A+DPTR  ;JUMP BACK INTO MAIN PROGRAM

```

This example illustrates several points:

- 1) The "subroutine" does not end with a normal return statement; instead, an indirect jump relative to the data pointer returns execution to the first instruction following the parameter list. The two initial POP instructions correct the stack pointer contents.
- 2) Either an ACALL or LCALL works with the subroutine, since each pushes the address of the *next* instruction or data byte onto the stack. The call may be made from anywhere in the full 8051 address space, since the MOVC instruction accesses all 64K bytes.

## MCS-51 APPLICATION EXAMPLES

- 3) The parameters passed to the utility can be listed in whatever order is most convenient, which may not be that in which they're used. The utility has essentially "random access" to the parameter list, by loading the appropriate constant into the accumulator before each MOVC instruction.
- 4) Other than the data pointer, the whole calling and processing sequence only affects the accumulator, PSW and pointer registers. The utility could have pushed these registers onto the stack (after popping the parameter list starting address), and popped before returning.

Passing parameters through in-line-code can be used in conjunction with other variable passing techniques.

The utility can also get input variables from working registers or from the stack, and return output variables to registers or to the stack.

### 9.1 PERIPHERAL INTERFACING TECHNIQUES

#### 9.1.1 I/O Port Reconfiguration (First Approach)

I/O ports must often transmit or receive parallel data in formats other than as eight-bit bytes. For example, if an application requires three five-bit latched output ports (called X, Y, and Z), these "virtual" ports could be mapped onto the pins of "physical" ports 1 and 2 (see example at bottom of page).

This pin assignment leaves P2.7 free for use as a test pin, input data pin, or control output through software.

Notice that the bits of port Z are reversed. The highest-order port Z pin corresponds to pin P2.2, and the lowest-order pin of port Z is P2.6, due to P.C. board layout considerations. When connecting an 8051 to an immediately adjacent keyboard column decoder or another device with weighted inputs, the corresponding pins may not be aligned. The interconnections must be "scrambled" to compensate either with interwoven circuit board traces or through software (as shown on the next page).

| PORT "Z" |      |      |      |      |      | PORT "Y" |      |      |      |      | PORT "X" |      |      |      |      |
|----------|------|------|------|------|------|----------|------|------|------|------|----------|------|------|------|------|
| -        | PZ0  | PZ1  | PZ2  | PZ3  | PZ4  | PY4      | PY3  | PY2  | PY1  | PY0  | PX4      | PX3  | PX2  | PX1  | PX0  |
| P2.7     | P2.6 | P2.5 | P2.4 | P2.3 | P2.2 | P2.1     | P2.0 | P1.7 | P1.6 | P1.5 | P1.4     | P1.3 | P1.2 | P1.1 | P1.0 |



## MCS-51 APPLICATION EXAMPLES

```

PX_MAP DATA 20H
PY_MAP DATA 21H
PZ_MAP DATA 22H
;
;
OUT_PX: ANL A,#00011111B ;CLEAR BITS ACC.7 - ACC. 5
        MOV PX_MAP,A ;SAVE DATA IN MAP BYTE
        ACALL OUT_P1 ;UPDATE PORT 1 OUTPUT LATCH
        RET
;
;
OUT_PY: MOV PY_MAP,A ;SAVE IN MAP BYTE
        ACALL OUT_P1 ;UPDATE PORT 1
        ACALL OUT_P2 ;AND PORT 2 OUTPUT LATCHES
        RET
;
;
OUT_PZ: MOV PZ_MAP,A ;SAVE DATA IN MAP BYTE
        ACALL OUT_P2 ;UPDATE PORT 2.
        RET
;
;
OUT_P1: MOV A,PY_MAP ;OUTPUT ALL P1 BITS
        SWAP A
        RL A ;SHIFT PY_MAP LEFT 5 BITS
        ANL A,#11100000B ;MASK OUT GARBAGE
        ORL A,PX_MAP ;INCLUDE PX_MAP BITS
        MOV P1,A
        RET
;
;
OUT_P2: MOV C,PZ_MAP.0 ;LOAD CY WITH P2.6 BIT
        RLC A ;AND SHIFT INTO ACC.
        MOV C,PZ_MAP.1 ;LOAD CY WITH P2.5 BIT
        RLC A ;AND SHIFT INTO ACC.
        MOV C,PZ_MAP.2 ;LOAD CY WITH P2.4 BIT
        RLC A ;AND SHIFT INTO ACC.
        MOV C,PZ_MAP.3 ;LOAD CY WITH P2.3 BIT
        RLC A ;AND SHIFT INTO ACC.
        MOV C,PZ_MAP.4 ;LOAD CY WITH P2.2 BIT
        RLC A ;AND SHIFT INTO ACC.
        MOV C,PZ_MAP.4 ;LOAD CY WITH P2.1 BIT
        RLC A ;AND SHIFT INTO ACC.
        MOV C,PZ_MAP.3 ;LOAD CY WITH P2.0 BIT
        RLC A ;AND SHIFT INTO ACC.
        SETB ACC.7 ;(ASSUMING INPUT ON P2.7)
        MOV P2,A
        RET

```

Writing to the virtual ports must not affect any other pins. Since the virtual output algorithms are non-trivial, a subroutine is needed for each port: OUT\_PX, OUT\_PY and OUT\_PZ. Each is called with data to output right-justified in the accumulator, and any data in bits ACC.7-ACC.5 is insignificant. Each subroutine saves the data in a "map" variable for the virtual port, then calls other subroutines which use the data in the various map bytes to compute and output the eight-bit pattern needed for each physical port affected. The two level structure of the above subroutines can be modified somewhat if code efficiency and execution speed are critical: incorporate the code shown as subroutines OUT\_P1 and OUT\_P2 directly into the code for OUT\_PX and OUT\_PZ, in place of the corresponding ACALL instructions. OUT\_PY would not be changed, but now the destinations for its ACALL instructions would be alternate entry points in OUT\_PX and OUT\_PZ, instead of isolated subroutines.

### 9.1.2 I/O Port Configuration (Second Approach)

A trickier situation arises if two sections of code which write to the same port or register, or call virtual output routines like those above, need to be executed at different interrupt levels. For example, suppose the background program wants to rewrite Port X (using the port associations in the previous example), and has computed the bit pattern needed for P1. An interrupt is detected just before the MOV

P1,A instruction, and the service routine tries to write Port Y. The service routine would correctly update P1 and P2, but upon returning to the background program P1 is immediately *re-written* with the data computed *before* the interrupt! Now pins P2.1 and P2.0 indicate (correctly) data written to port Y in the interrupt routine, but the earlier data written to P1.7-P1.5 is no longer valid. The same sort of confusion could arise if a high-level interrupt disrupted such an output sequence.

One solution is to disable interrupts around any section of code which must not be interrupted (called a "critical section"), but this would adversely affect interrupt latency. Another is to have interrupt routines set or clear a flag ("semaphore") when a common resource is altered — a rather complex and elaborate system.

An easier way to ensure that any instruction which writes the port X field of P1 does not change the port Y field pins from their state *at the beginning of that instruction*, is shown next. A number of 8051 operations read, modify, and write the output port latches all in one instruction. These are the arithmetic and logical instructions (INC, DEC, ANL, ORL, etc.), where an addressed byte is both the destination variable and one of the source operands. Using these instructions, instead of data moves, eliminates the critical section problem entirely.



## MCS-51 APPLICATION EXAMPLES

```

OUT_PX: ANL     P1,#11100000B    ;CLEAR BITS P1.4 · P1.0
        ORL     P1,A             ;SET P1 PIN FOR EACH ACC BIT SET
        RET
;
;
;
OUT_PY: MOV     B,#20H
        MUL     AB               ;SHIFT B A LEFT 5 BITS.
        ANL     P1,#00011111B    ;CLEAR PY FIELD OF PORT 1
        ORL     P1,A             ;SET PY BITS ON PORT 1
        MOV     A,B             ;LOAD 2 BITS SHIFTED INTO B
        ANL     P2,#11111100B    ;AND UPDATE P2
        ORL     P2,A
        RET
;
;
;
OUT_PZ: RRC     A               ;MOVE ORIGINAL ACC.0 INTO CY
        MOV     P2.6,C          ;AND STORE TO PIN P2.6.
        RRC     A               ;MOVE ORIGINAL ACC.1 INTO CY
        MOV     P2.5,C          ;AND STORE TO PIN P2.5.
        RCC     A               ;MOVE ORIGINAL ACC.2 INTO CY
        MOV     P2.4,C          ;AND STORE TO PIN P2.4.
        RRC     A               ;MOVE ORIGINAL ACC.3 INTO CY
        MOV     P2.3,C          ;AND STORE TO PIN P2.3.
        RRC     A               ;MOVE ORIGINAL ACC.4 INTO CY
        MOV     P2.2,C          ;AND STORE TO PIN P2.2.
        RET

```

### 9.1.3 8243 Interfacing

The 8051's quasi-bidirectional port structure lets each I/O pin input data, output data, or serve as a test pin or output strobe under software control. An example of these modes operating in conjunction is the host-processor

interface expected by an 8243 I/O expander. Even though the 8051 does not include 8048-type instructions for interfacing with an 8243, the parts can be interconnected and the protocol may be emulated with simple software; see Figure 9-2.

```

;
;IN8243  INPUT DATA FROM AN 8243 I/O EXPANDER
;
;        CONNECTED TO P23-P20.
;
;        P25 & P24 MIMIC CS & PROG.
;
;        P27-P26 USED AS INPUTS. CODE FOR
;
;        PORT TO BE READ IN ACC.1-ACC.0
;
;

```

## MCS-51 APPLICATION EXAMPLES

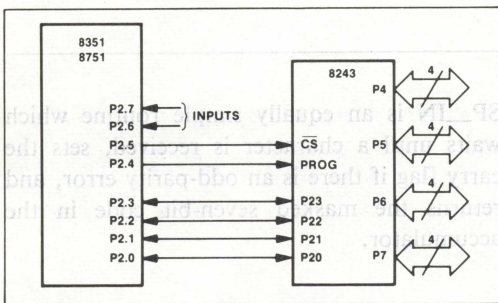
|                |            |                      |   |
|----------------|------------|----------------------|---|
| <b>PROG</b>    | <b>BIT</b> | <b>P2.4</b>          | <b>;SYMBOLIC PIN DESCRIPTION</b>            |
| <b>IN8243:</b> | <b>ORL</b> | <b>A,#11010000B</b>  | <b>;SET PROG AND PINS USED AS INPUT</b>     |
|                | <b>MOV</b> | <b>P2,A</b>          | <b>;OUTPUT PORT CODE AND OPERATION CODE</b> |
|                | <b>CLR</b> | <b>PROG</b>          | <b>;LOWER PROG TO LATCH ADDRESS</b>         |
|                | <b>ORL</b> | <b>P2,#00001111B</b> | <b>;SET LOW ORDER PINS FOR INPUT</b>        |
|                | <b>MOV</b> | <b>A,P2</b>          | <b>;READ IN PORT DATA</b>                   |
|                | <b>ORL</b> | <b>P2,#00110000B</b> | <b>;SET PROG AND CS HIGH</b>                |

### 9.1.4 Software Delay Timing

Many 8051 applications involve exact control over output timing. A software-generated output strobe, for instance, might have to be *exactly* 50  $\mu$ sec. wide. The DJNZ operation can insert a one instruction software delay into

a piece of code, adding a moderate time delay of two instruction cycles per iteration. For example, two instructions can add a 49- $\mu$ sec. software delay loop to code to generate a pulse on the WR pin.

|             |               |
|-------------|---------------|
| <b>CLR</b>  | <b>WR</b>     |
| <b>MOV</b>  | <b>R2,#24</b> |
| <b>DJNZ</b> | <b>\$2,\$</b> |
| <b>SETB</b> | <b>WR</b>     |



**Figure 9-2 Connecting an 8051 with an 8243 I/O Expander**

The dollar sign in this example is a special character meaning "the address of this instruction." It can be used to eliminate instruction labels on nearby source lines.

### 9.1.5 Serial Port and Timer Configuration

Configuring the 8051's Serial Port for a given data rate and protocol requires essentially three short sections of software. On power-up or hardware reset the serial port and timer

control words must be initialized to the appropriate values. Additional software is also needed in the transmit routine to load the serial port data register and in the receive routine to unload the data as it arrives.

To choose one arbitrary example, assume the 8051 should communicate with a standard CRT operating at 2400 baud (bits per second). Each character is transmitted as seven data bits, odd parity, and one stop bit. The resulting character rate is 2400 baud/9 bits, approximately 265 characters per second.

For the sake of clarity, the transmit and receive subroutines here are driven by simple-minded software status polling code rather than interrupts. The serial port must be initialized to 8-bit UART mode (SM0, SM1 = 01), enabled to receive all messages (SM2 = 0, REN = 1). The flag indicating that the transmit register is free for more data will be artificially set in order to let the output software know the output register is available. All this can be set up with instruction at label SPINIT.



## MCS-51 APPLICATION EXAMPLES

Timer 1 will be used in auto-reload mode as a baud rate generator. To achieve a data rate of 2400 baud, the timer must divide the 1MHz internal clock by

$$\frac{1 \times 10^6}{(32)(2400)}$$

which equals 13 (actually, 13.02) instruction cycles. The timer must reload the value -13, or 0F3H, as shown by the code at label TIINIT. (ASM51 will accept both the signed decimal or hexadecimal representations.)

```

;
;      INITIALIZE SERIAL PORT
;      FOR 8-BIT UART MODE
;      & SET TRANSMIT READY FLAG.
SPINIT:  MOV     SCON,#01010010B
;
;      INITIALIZE TIMER 1 FOR
;      AUTO-RELOAD AT 32 X 2400HZ
;      (T0 USED AS GATED 16-BIT COUNTER.)
;THINIT: MOV     TCON,#11010010B
;      MOV     TH1,# 13
;      SETB    TR1
;
;

```

### 9.1.6 Simple Serial I/O Drivers

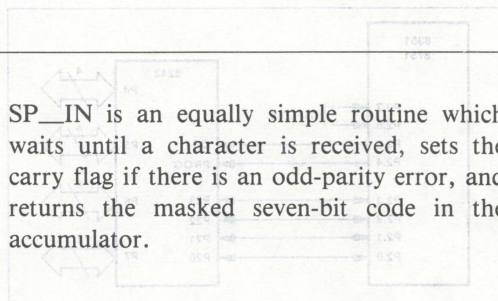
SP\_OUT is a simple subroutine to transmit the character passed to it in the accumulator. First it must compute the parity bit, insert it into the data byte, wait until the transmitter is available, output the character, and then return.

```

;SP_OUT ADD ODD PARITY TO ACC AND
;TRANSMIT WHEN SERIAL PORT READY
;
SP_OUT: MOV C,P
CPL C
MOV ACC.7,C
JNB TI,$
CLR TI
MOV SBUF,A
RET

```

SP\_IN is an equally simple routine which waits until a character is received, sets the carry flag if there is an odd-parity error, and returns the masked seven-bit code in the accumulator.



## MCS-51 APPLICATION EXAMPLES

```

;SP_IN INPUT NEXT CHARACTER FROM SERIAL PORT.
;      SET CARRY IF ODD-PARITY ERROR
;

```

```

SP_IN:  JNB  RI,$
        CLR  RI
        MOV  A,SBUF
        MOV  C,P
        CPL  C
        ANL  A,#7FH
        RET

```

### 9.1.7 Transmitting Serial Port Character Strings

Any application which transmits characters through a serial port to an ASCII output device will on occasion need to output

“canned” messages, including error messages, diagnostics, or operator instructions. These character strings are most easily defined with in-line data bytes defined with the DB directive.

|          |      |                           |                                  |
|----------|------|---------------------------|----------------------------------|
| CR       | EQU  | 0DH                       | ;ASCII CARRIAGE RET              |
| LF       | EQU  | 0AH                       | ;ASCII LINE-FEED                 |
| ESC      | EQU  | 1BH                       | ;ASCII ESCAPE CODE               |
| ;        | ...  | .....                     | ;                                |
|          | CALL | XSTRING                   |                                  |
|          | DB   | CR,LF                     | ;NEW LINE                        |
|          | DB   | 'INTEL DELIVERS'          | ;MESSAGE                         |
|          | DB   | ESC                       | ;ESCAPE CHARACTER                |
| ;        |      |                           |                                  |
| ;        |      | (CONTINUATION OF PROGRAM) |                                  |
| ;        |      |                           |                                  |
| ;        |      |                           |                                  |
| XSTRING: | ...  | .....                     | ;LOAD DPTR WITH FIRST CHARACTER  |
|          | POP  | DPH                       |                                  |
|          | POP  | DPL                       |                                  |
| XSTR_1:  | CLR  | A                         | ;(ZERO OFFSET)                   |
|          | MOVC | A,@A + DPTR               | ;FETCH FIRST CHARACTER OF STRING |
| XSTR_2:  | JNB  | TI,\$                     | ;WAIT UNTIL TRANSMITTER READY    |
|          | CLR  | TI                        | ;MARK AS NOT READY               |
|          | MOV  | SBUF,A                    | ;OUTPUT NEXT CHARACTER           |
|          | INC  | DPTR                      | ;BUMP POINTER                    |
|          | CLR  | A                         |                                  |
|          | MOVC | A,@A + DPTR               | ;GET NEXT OUTPUT CHARACTER       |
|          | CJNE | A,#ESC,XSTR_2             | ;LOOP UNTIL ESCAPE READ          |
|          | MOV  | A,#1                      |                                  |
|          | JMP  | @A + DPTR                 | ;RETURN TO CODE AFTER ESCAPE     |



## MCS-51 APPLICATION EXAMPLES

### 9.1.8 Recognizing and Processing Special Cases

Before operating on the data it receives, a subroutine might give “special handling” to certain input values. Consider a word processing device which receives ASCII characters through the 8051 serial port and drives a thermal hard-copy printer. A standard routine translates most printing characters to bit pat-

tens, but certain control characters (<DEL>, <CR>, <LF>, <BEL>, <ESC>, or <SP>) must invoke corresponding special routines. Any other character with an ASCII code less than 20H should be translated into the <NUL> value, 00H, and processed with the printing characters. The CJNE operation provides essentially a one-instruction CASE statement.

|                |             |                         |  |
|----------------|-------------|-------------------------|--|
| <b>CHAR</b>    | <b>EQU</b>  | <b>R7</b>               | <b>;CHARACTER CODE VARIABLE</b>          |
| <b>INTERP:</b> | <b>CJNE</b> | <b>CHAR,#7FH,INTP_1</b> | <b>;SKIP UNLESS RUBOUT</b>               |
|                | <b>...</b>  | <b>.....</b>            | <b>(SPECIAL ROUTINE FOR RUBOUT CODE)</b> |
|                | <b>RET</b>  |                         |  |
| <b>INTP_1:</b> | <b>CJNE</b> | <b>CHAR,#07H,INTP_2</b> | <b>;SKIP UNLESS BELL</b>                 |
|                | <b>...</b>  | <b>.....</b>            | <b>(SPECIAL ROUTINE FOR BELL CODE)</b>   |
|                | <b>RET</b>  |                         |  |
| <b>INTP_2:</b> | <b>CJNE</b> | <b>CHAR,#0AH,INTP_3</b> | <b>;SKIP UNLESS LFEED</b>                |
|                | <b>...</b>  | <b>.....</b>            | <b>(SPECIAL ROUTINE FOR LFEED CODE)</b>  |
|                | <b>RET</b>  |                         |  |
| <b>INTP_3:</b> | <b>CJNE</b> | <b>CHAR,#0DH,INTP_4</b> | <b>;SKIP UNLESS RETURN</b>               |
|                | <b>...</b>  | <b>.....</b>            | <b>(SPECIAL ROUTINE FOR RETURN CODE)</b> |
|                | <b>RET</b>  |                         |  |
| <b>INTP_4:</b> | <b>CJNE</b> | <b>CHAR,#1BH,INTP_5</b> | <b>;SKIP UNLESS ESCAPE</b>               |
|                | <b>...</b>  | <b>.....</b>            | <b>(SPECIAL ROUTINE FOR ESCAPE CODE)</b> |
|                | <b>RET</b>  |                         |  |
| <b>INTP_5:</b> | <b>CJNE</b> | <b>CHAR,#20H,INTP_6</b> | <b>;SKIP UNLESS SPACE</b>                |
|                | <b>...</b>  | <b>.....</b>            | <b>(SPECIAL ROUTINE FOR SPACE CODE)</b>  |
|                | <b>RET</b>  |                         |  |
| <b>INTP_6:</b> | <b>JC</b>   | <b>PRINTC</b>           | <b>;JUMP IF CODE 20 H</b>                |
|                | <b>MOV</b>  | <b>CHAR,#0</b>          | <b>;REPLACE CONTROL CHARACTER WITH</b>   |
|                |             |                         | <b>;NULL CODE</b>                        |
| <b>PRINTC:</b> |             |                         | <b>;PROCESS STANDARD PRINTING</b>        |
|                | <b>...</b>  | <b>.....</b>            | <b>;CHARACTER</b>                        |
|                | <b>RET</b>  |                         |  |

### 9.1.9 Synchronizing Timer Overflows

8051 timer overflows automatically generate an internal interrupt request, which will vector program execution to the appropriate interrupt service routine if interrupts are enabled and no other service routines are in progress at the time. However, it is not predictable *exactly* how long it will take to reach the service routine. The service routine call takes two instruction cycles, but 1, 2, or 4 additional cycles may be needed to complete the instruction in progress. If the background program ever disables interrupts, the response latency could further increase by a few instruction cycles. (Critical sections generally involve simple instruction sequences — rarely multiplies or divides.) Interrupt response delay is generally negligible, but certain time-critical application must take the exact delay into account. For example, generating interrupts with timer 1 every millisecond (1000 instruction cycles) or so

would normally call for reloading it with the value  $-1000$  (0FC30H). But if the interrupt interval (average over time) must be accurate to 1 instruction cycle, the 16-bit value reload into the timer must be computed, taking into account when the timer actually overflowed.

This simply requires reading the appropriate timer, which has been incremented each cycle since the overflow occurred. A sequence like the one below can stop the timer, compute how much time should elapse before the next interrupt, and reload and restart the timer. The double-precision calculation shown here compensates for any amount of timer overrun within the maximum interval. Note that it also takes into account that the timer is stopped for seven instruction cycles in the process. All interrupts are disabled, so a higher priority request will not be able to disrupt the time-critical code section.

|   |      |                        |                               |
|---|------|------------------------|-------------------------------|
| ; | ...  | .....                  |                               |
|   | CLR  | EA                     | ;DISABLE ALL INTERRUPTS       |
|   | CLR  | TR1                    | ;STOP TIMER 1                 |
|   | MOV  | A,#LOW( $-1000 + 7$ )  | ;LOAD LOW-ORDER DESIRED COUNT |
|   | ADD  | A,TL1                  | ;CORRECT FOR TIMER OVERRUN    |
|   | MOV  | TL1,A                  | ;RELOAD LOW-ORDER BYTE.       |
|   | MOV  | A,#HIGH( $-1000 + 7$ ) | ;REPEAT FOR HIGH-ORDER BYTE.  |
|   | ADDC | A,TH1                  |                               |
|   | MOV  | TH1,A                  |                               |
|   | SETB | TR1                    | ;RESTART TIMER                |
| ; | ...  | .....                  |                               |



### 9.1.10 Reading a Timer/Counter "On-the-Fly"

The preceding example simply stopped the timer before changing its contents. This is normally done when reloading a timer so that the time at which the timer is started (i.e. the "run" flag is set) can be exactly controlled. There are situations, though, when it is desired to read the current count without disrupting the timing process. The 8051 timer/counter registers can all be read or written while they are running, but a few precautions must be taken.

Suppose the subroutine RDTIME should

return in  $\langle R1 \rangle$ ,  $\langle R0 \rangle$  a sixteen-bit value indicating the count in timer 0. The instant at which the count was sampled is not as critical as the fact that the value returned must have been valid at some point while the routine was in progress. There is a potential problem that between reading the two halves, a low-order register overflow might increment the high-order register, and the two data bytes returned would be "out of phase." The solution is to read the high-order byte first, then the low-order byte, and then confirm that the high-order byte has not changed. If it has, repeat the whole process.

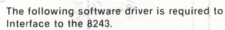
```
RDTIME:  MOV    A,TH0    ;SAMPLE TIMERO (HIGH)
          MOV    R0,TL0    ;SAMPLE TIMERO (LOW)
          CJNE   A,TH0,RDTIME ;REPEAT IF NECESSARY
          MOV    R1,A      ;STORE VALID READ
          RET
```

## 9.2 CONNECTING TO PERIPHERALS

This section shows in very general terms some basic circuits for expanding the 8051 Family. The schematics included in this chapter should give the designer and insight into connecting external peripherals and memories to the 8051.

```
...
CLR      EA
CLR      TRI
MOV      A,LOW(-1000 + ?)
ADD      A,TL1
MOV      TL1,A
MOV      A,HIGH(-1000 + ?)
ADDC     A,TH1
MOV      TH1,A
SETB     TRI
...
```

THE UNIVERSITY OF CHICAGO



### Mixing Parallel Output, Input, and

```

IN8243      INPUT DATA FROM AN 8243 I/O EXPANDER
             CONNECT TO P23:P20
             P25 & P24 MIMIC CS/ & PROG
             P27-P26 USED AS INPUTS
             PORT TO BE READ IN ACC

```

```
IN8243:  ORL    A,#11010000B
          MOV   P2,A      ;OUTPUT INSTRUCTION CODE
          CLR   P2,A      ;FALLING EDGE OF PROG
          ORL   P2,#00001111B ;SET FOR INPUT
          MOV   A,P2      ;READ INPUT DATA
          SETB  P2,A      ;RETURN PROG HIGH
```

**Figure 9-3. I/O Expansion Using an 8243**



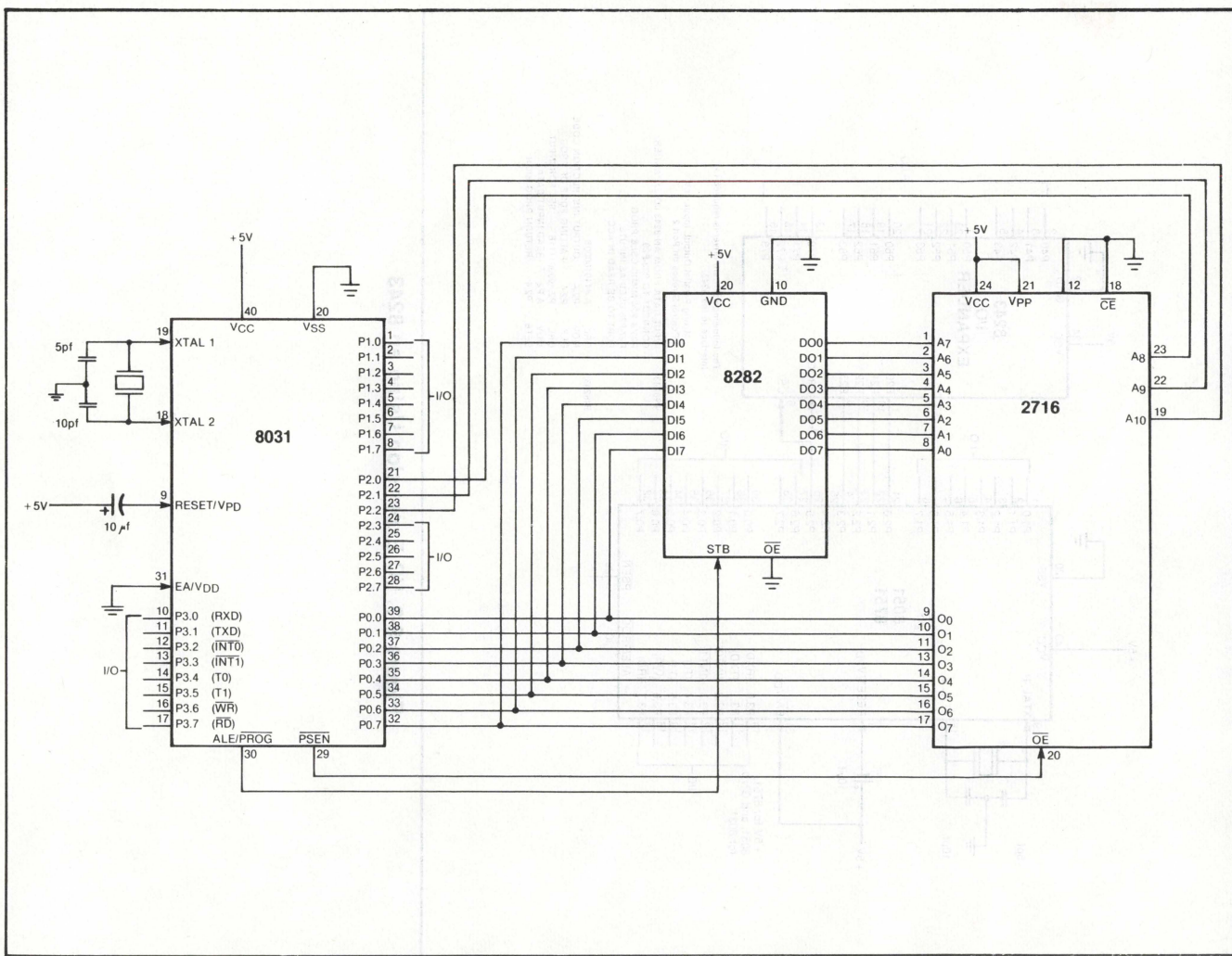


Figure 9-4. External Program Memory Using a 2716



### Figure 9-5. External Program Memory Using a 2732



## MCS-51 INSTRUCTION SET

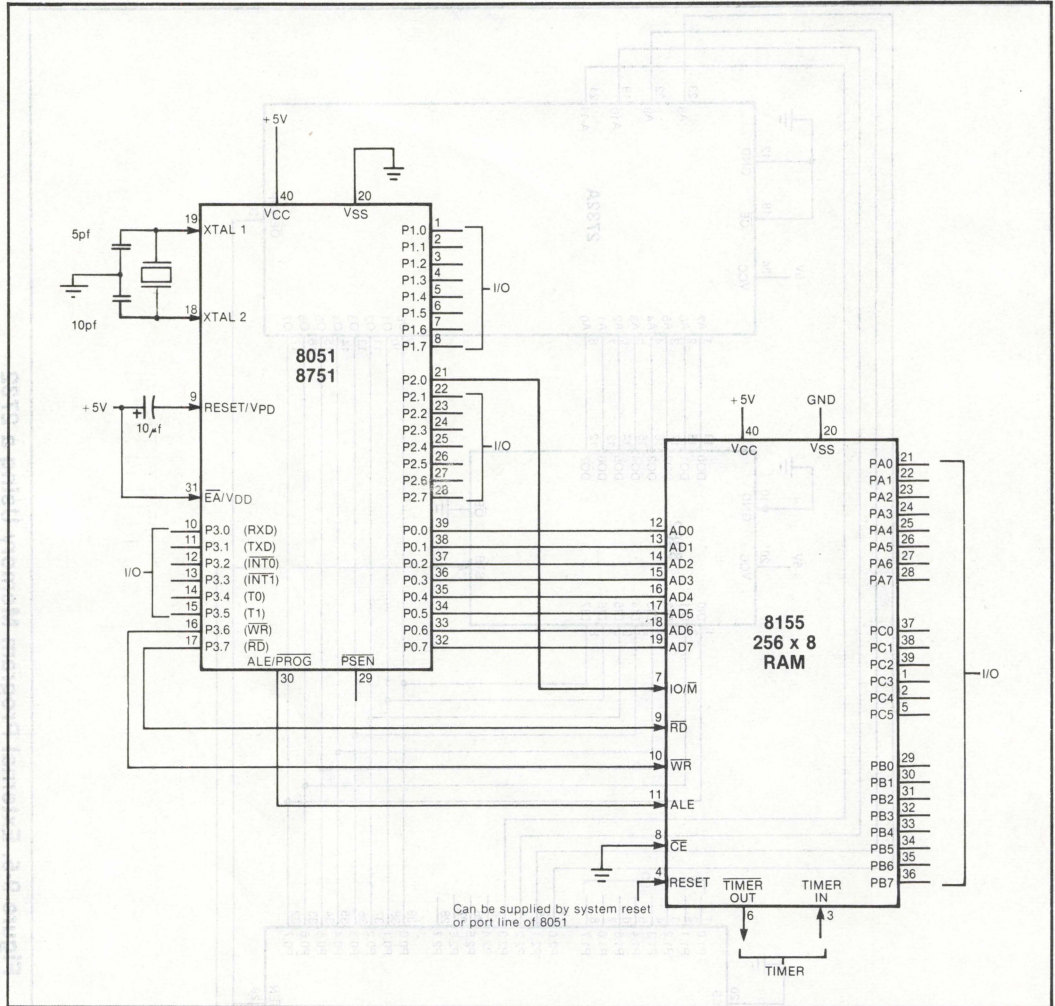


Figure 9-6. Adding a Data Memory and I/O Expander

# MCS-51 INSTRUCTION SET

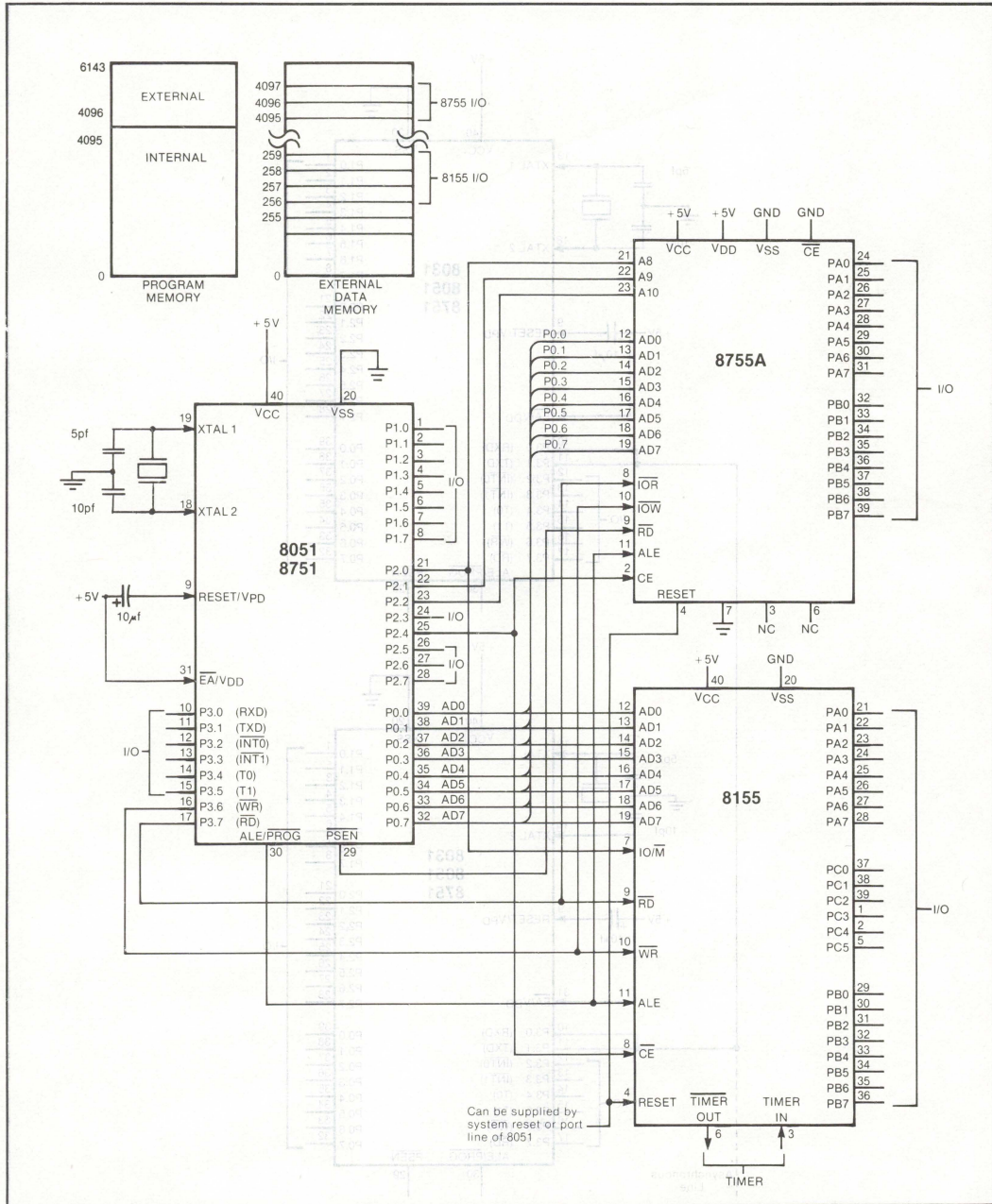


Figure 9-7. The Three-Chip System



The diagrams illustrate the connection of an 8031 microcontroller to an 8051 microcontroller. The top diagram shows a synchronous connection, and the bottom diagram shows an asynchronous connection. Both diagrams include power supply connections (VCC, VSS, +5V), reset connections (RESET/VPD), and I/O connections (P0-P7, P2.0-P2.7).

**Top Diagram (Synchronous Connection):**

- 8031 Microcontroller:**
  - VCC: 40
  - VSS: 20
  - XTAL 1: 19
  - XTAL 2: 18
  - RESET/VPD: 9
  - EA/VDD: 31
  - P3.0 (RXD): 10
  - P3.1 (TXD): 11
  - P3.2 (INT0): 12
  - P3.3 (INT1): 13
  - P3.4 (T0): 14
  - P3.5 (T1): 15
  - P3.6 (WR): 16
  - P3.7 (RD): 17
  - ALE/PROG: 30
  - PSEN: 29
- 8051 Microcontroller:**
  - P1.0-P1.7: 1-8
  - P2.0-P2.7: 21-28
  - P0.0-P0.7: 39-32
- Connections:**
  - 8031 ALE/PROG (30) to 8051 PSEN (29)
  - 8031 PSEN (29) to 8051 P0.0 (39)

**Bottom Diagram (Asynchronous Connection):**

- 8031 Microcontroller:**
  - VCC: 40
  - VSS: 20
  - XTAL 1: 19
  - XTAL 2: 18
  - RESET/VPD: 9
  - EA/VDD: 31
  - P3.0 (RXD): 10
  - P3.1 (TXD): 11
  - P3.2 (INT0): 12
  - P3.3 (INT1): 13
  - P3.4 (T0): 14
  - P3.5 (T1): 15
  - P3.6 (WR): 16
  - P3.7 (RD): 17
  - ALE/PROG: 30
  - PSEN: 29
- 8051 Microcontroller:**
  - P1.0-P1.7: 1-8
  - P2.0-P2.7: 21-28
  - P0.0-P0.7: 39-32
- Connections:**
  - 8031 ALE/PROG (30) to 8051 PSEN (29) through a 10k resistor
  - 8031 PSEN (29) to 8051 P0.0 (39)

## MCS-51 INSTRUCTION SET

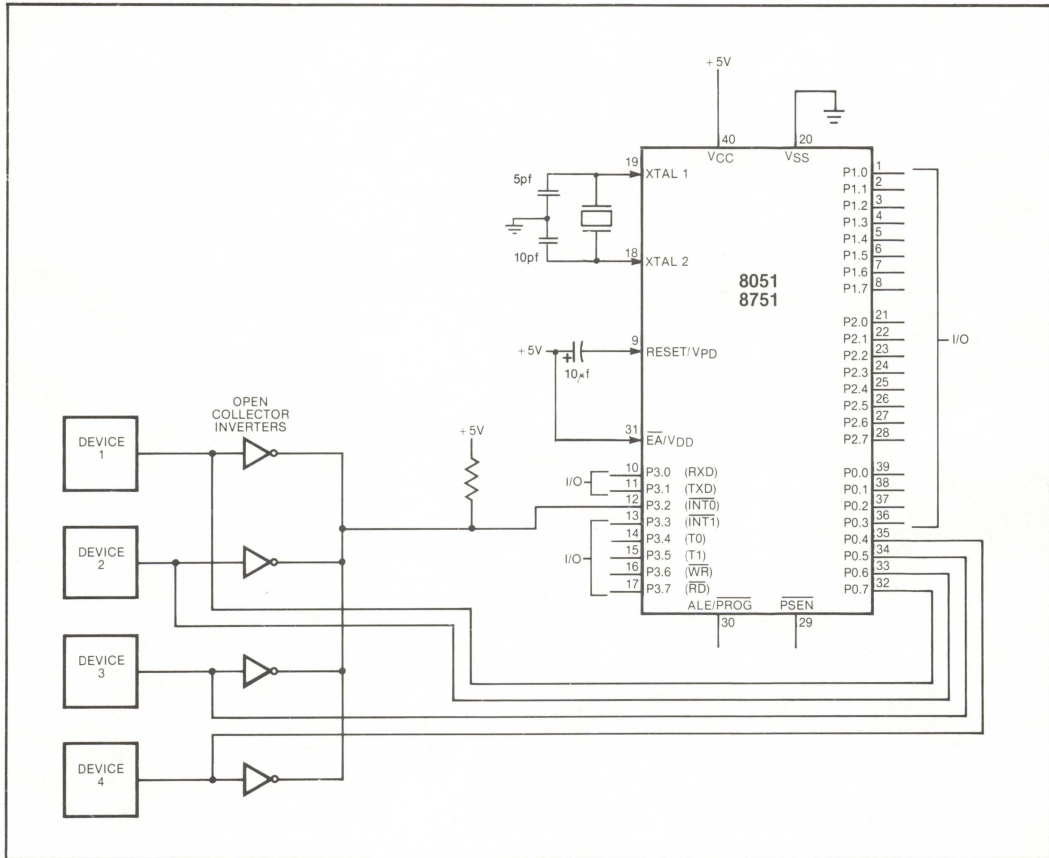


Figure 9-9. Multiple Interrupt Sources



---

# Microcontroller Specifications

---

10

# HMOS SINGLE-COMPONENT 8-BIT MICROCOMPUTER

- 8-bit CPU, ROM, RAM, I/O in Single 20-pin Dip
- Single +5V Supply (+4.5V to 6.5V)
- 8.38  $\mu$ sec Cycle with 3.58 MHz XTAL. All instructions 1 or 2 cycles.
- Instructions — 8048 Subset
- Zero-Cross Detection Capability
- 1K x 8 ROM
- 64 x 8 RAM
- 13 I/O Lines
- Internal Timer/Counter
- 30mA Operation @ 25°C
- Clock Generated with Single Inductor, Resistor or Crystal

The Intel 8020H is a 20-pin DIP version of the Intel 8021H. This totally self-sufficient 8-bit parallel computer is fabricated on a single silicon chip using Intel's advanced N-channel silicon gate HMOS process.

The 8020H is designed for high-volume cost-sensitive applications such as home entertainment products, appliances and simple control tasks. In addition to its significant board real estate savings, the 8020H has a subset of the industry standard 8048 instruction set optimized for byte efficiency and control.

The 8020H also features a 1K x 8 program memory, 64 x 8 data memory, 13 I/O lines and an 8-bit timer/counter with on-board oscillator and clock circuits. Standard low cost TTL can be used to expand the number of I/O lines.

To minimize development problems and maximize flexibility, an 8020H system can be easily designed using the EM-2 emulator board and an 8020H pin scrambler interface.

For a complete summary of 8020H operating characteristics, refer to the 8021H data sheet.

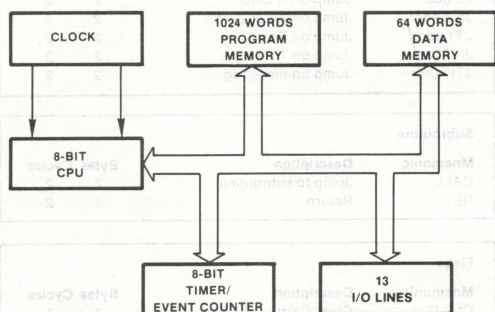


Figure 1.  
Block Diagram

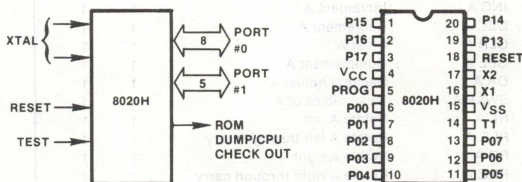


Figure 2.  
Logic Symbol

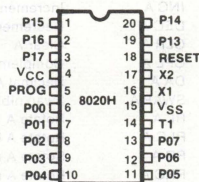


Figure 3. Pin  
Configuration



Table 1. Pin Description

| Symbol            | Pin No.      | Function  |
|-------------------|--------------|---|
| V <sub>SS</sub>   | 15           | Circuit GND potential   |
| V <sub>CC</sub>   | 4            | +5V power supply  |
| PROG              | 5            | Output pin used to dump internal ROM and to activate CPU checkout.  |
| P00-P07<br>Port 0 | 6-13         | 8-bit quasi-bidirectional port  |
| P3-P17<br>Port 1  | 19-20<br>1-3 | 5-bit quasi-bidirectional port  |
| T1                | 14           | Input pin testable using the JT1 and JNT1 instructions. Can be designated the timer/event counter input using the STRT CNT instruction. Also allows zero-crossover sensing of slowly moving AC inputs |
| RESET             | 18           | Input used to initialize the processor by clearing status flip-flops and setting program counters to zero. (Active High).   |
| XTAL1             | 16           | One side of crystal or inductor input for internal oscillator. Also input for external source. (Not TTL compatible.)  |
| XTAL2             | 17           | Other side of timing control element.   |

Table 2. Instruction Set

| Accumulator    |   |       |        |
|----------------|---|-------|--------|
| Mnemonic       | Description                               | Bytes | Cycles |
| ADD A, R       | Add register to A                         | 1     | 1      |
| ADD A, @R      | Add data memory to A                      | 1     | 1      |
| ADD A, # data  | Add immediate to A                        | 2     | 2      |
| ADDC A, R      | Add with carry                            | 1     | 1      |
| ADDC A, @R     | Add with carry                            | 1     | 1      |
| ADDC A, # data | Add with carry                            | 2     | 2      |
| ANL A, R       | And register to A                         | 1     | 1      |
| ANL A, @R      | And data memory to A                      | 1     | 1      |
| ANL A, # data  | And immediate to A                        | 2     | 2      |
| ORL A, R       | Or register to A                          | 1     | 1      |
| ORL A, @R      | Or data memory to A                       | 1     | 1      |
| ORL A, # data  | Or immediate to A                         | 2     | 2      |
| XRL A, R       | Exclusive or register to A                | 1     | 1      |
| XRL A, @R      | Exclusive or data memory to A             | 1     | 1      |
| XRL A, # data  | Exclusive or immediate to A               | 2     | 2      |
| INC A          | Increment A                               | 1     | 1      |
| DEC A          | Decrement A                               | 1     | 1      |
| CLR A          | Clear A                                   | 1     | 1      |
| CPL A          | Complement A                              | 1     | 1      |
| DA A           | Decimal Adjust A                          | 1     | 1      |
| SWAP A         | Swap nibbles of A                         | 1     | 1      |
| RL A           | Rotate A left                             | 1     | 1      |
| RLC A          | Rotate A left through carry               | 1     | 1      |
| RR A           | Rotate A right                            | 1     | 1      |
| RRC A          | Rotate A right through carry              | 1     | 1      |
| Registers      |   |       |        |
| Mnemonic       | Description                               | Bytes | Cycles |
| INC R          | Increment register                        | 1     | 1      |
| INC @R         | Increment data memory                     | 1     | 1      |
| Branch         |   |       |        |
| Mnemonic       | Description                               | Bytes | Cycles |
| JMP addr       | Jump unconditional                        | 2     | 2      |
| JMPP @A        | Jump indirect                             | 1     | 2      |
| DJNZ R, addr   | Decrement register and Jump on R not zero | 2     | 2      |
| JC addr        | Jump on Carry = 1                         | 2     | 2      |
| JNC addr       | Jump on Carry = 0                         | 2     | 2      |
| JZ addr        | Jump on A Zero                            | 2     | 2      |
| JNZ addr       | Jump on A not Zero                        | 2     | 2      |
| JT1 addr       | Jump on T1 = 1                            | 2     | 2      |
| JNT1 addr      | Jump on T1 = 0                            | 2     | 2      |
| JTF addr       | Jump on timer flag                        | 2     | 2      |
| Subroutine     |   |       |        |
| Mnemonic       | Description                               | Bytes | Cycles |
| CALL           | Jump to subroutine                        | 2     | 2      |
| RET            | Return                                    | 1     | 2      |
| Flags          |   |       |        |
| Mnemonics      | Description                               | Bytes | Cycles |
| CLR C          | Clear Carry                               | 1     | 1      |
| CPL C          | Complement Carry                          | 1     | 1      |
| Input/Output   |   |       |        |
| Mnemonic       | Description                               | Bytes | Cycles |
| IN A, P        | Input port to A                           | 1     | 2      |
| OUTL P, A      | Output A to port                          | 1     | 2      |

Table 2. Instruction Set Summary (cont.)

| Data Moves     |                                   |       |        |
|----------------|-----------------------------------|-------|--------|
| Mnemonics      | Description                       | Bytes | Cycles |
| MOV A, R       | Move register to A                | 1     | 1      |
| MOV A, @R      | Move data memory to A             | 1     | 1      |
| MOV A, # data  | Move immediate to A               | 2     | 2      |
| MOV R, A       | Move A to register                | 1     | 1      |
| MOV @R, A      | Move A to data memory             | 1     | 1      |
| MOV R, # data  | Move immediate to register        | 2     | 2      |
| MOV @R, # data | Move immediate to data memory     | 2     | 2      |
| XCH A, R       | Exchange A and register           | 1     | 1      |
| XCH A, @R      | Exchange A and data memory        | 1     | 1      |
| XCHD A, @R     | Exchange nibble of A and register | 1     | 1      |
| MOVP A, @A     | Move to A from current page       | 1     | 2      |

| Timer/Counter |                    |       |        |
|---------------|--------------------|-------|--------|
| Mnemonic      | Description        | Bytes | Cycles |
| MOV A, T      | Read Timer/Counter | 1     | 1      |
| MOV T, A      | Load Timer/Counter | 1     | 1      |
| STRT T        | Start Timer        | 1     | 1      |
| STRT CNT      | Start Counter      | 1     | 1      |
| STOP TCNT     | Stop Timer/Counter | 1     | 1      |

| Mnemonics | Description  | Bytes | Cycles |
|-----------|--------------|-------|--------|
| NOP       | No Operation | 1     | 1      |



## HMOS SINGLE-COMPONENT 8-BIT MICROCOMPUTER

- 8-Bit CPU, ROM, RAM, I/O in Single 28-Pin Package
- Single 5V Supply (+4.5V to 6.5V)
- 8.38  $\mu$ sec Cycle with 3.58 MHz XTAL; All instructions 1 or 2 Cycles
- 30mA Operation @ 25°C
- Instructions—8048 Subset
- High Current Drive Capability—2 Pins
- 1K x 8 ROM
- 64 x 8 RAM
- 21 I/O Lines
- Interval Timer/Event Counter
- Clock Generated with Single Inductor, Resistor or Crystal
- Zero-Cross Detection Capability
- Easily Expandable I/O

The Intel® 8021H is a totally self-sufficient 8-bit parallel computer fabricated on a single silicon chip using Intel's advanced N-channel silicon gate HMOS process. The features of the 8021H include a subset of the 8048 features optimized for low cost, high volume applications, plus additional I/O flexibility and power.

The 8021H contains 1K x 8 program memory, 64 x 8 data memory, 21 I/O lines, and an 8-bit timer/event counter, in addition to on-board oscillator and clock circuits. For systems that require extra I/O capability, the 8021H can be expanded using the 8243 or discrete logic.

This microcomputer is designed to be an efficient controller as well as an arithmetic processor. The 8021H has bit handling capability as well as facilities for both binary and BCD arithmetic. Efficient use of program memory results from an instruction set consisting mostly of single-byte instructions and no instructions over two bytes in length.

To minimize development problems and maximize flexibility, an 8021H system can be easily designed using the 8022 emulation board, the EM-2.

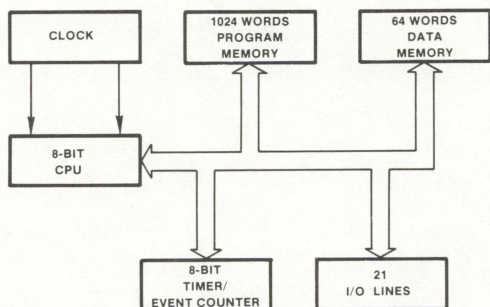


Figure 1.  
Block Diagram

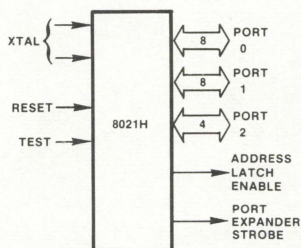


Figure 2.  
Logic Symbol

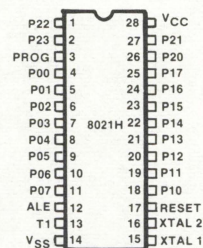


Figure 3. Pin  
Configuration



Table 1. Pin Description

| Symbol            | Pin No.      | Function  |
|-------------------|--------------|---|
| V <sub>SS</sub>   | 14           | Circuit GND potential   |
| V <sub>CC</sub>   | 28           | +5V power supply  |
| PROG              | 3            | Output Strobe for 8243 I/O Expander   |
| P00-P07<br>Port 0 | 4-11         | 8-bit quasi-bidirectional port  |
| P10-P17<br>Port 1 | 18-25        | 8-bit quasi-bidirectional port  |
| P20-P23<br>Port 2 | 26-27<br>1-2 | 4-bit quasi-bidirectional port P20-P23 also serve as a 4-bit I/O expander bus for 8243  |
| T1                | 13           | Input pin testable using the JT1 and JNT1 instructions. Can be designated the timer/event counter input using the STRT CNT instructions. Also allows zero-crossover sensing of slowly moving AC inputs. |
| RESET             | 17           | Input used to initialize the processor by clearing status flip-flops and setting program counters to zero. (Active high).   |
| ALE               | 12           | Address Latch Enable. Signal occurring once every 30 input clocks, used as an output clock.   |
| XTAL1             | 15           | One side of crystal resistor or inductor input for internal oscillator. Also input for external source. (Not TTL compatible.)   |
| XTAL2             | 16           | Other side of timing control element.   |

Table 2. Instruction Set Summary

| Accumulator    |                               |       |        |
|----------------|-------------------------------|-------|--------|
| Mnemonic       | Description                   | Bytes | Cycles |
| ADD A, R       | Add register to A             | 1     | 1      |
| ADD A, @R      | Add data memory to A          | 1     | 1      |
| ADD A, # data  | Add immediate to A            | 2     | 2      |
| ADDC A, R      | Add with carry                | 1     | 1      |
| ADDC A, @R     | Add with carry                | 1     | 1      |
| ADDC A, # data | Add with carry                | 2     | 2      |
| ANL A, R       | And register to A             | 1     | 1      |
| ANL A, @R      | And data memory to A          | 1     | 1      |
| ANL A, # data  | And immediate to A            | 2     | 2      |
| ORL A, R       | Or register to A              | 1     | 1      |
| ORL A, @R      | Or data memory to A           | 1     | 1      |
| ORL A, # data  | Or immediate to A             | 2     | 2      |
| XRL A, R       | Exclusive or register to A    | 1     | 1      |
| XRL A, @R      | Exclusive or data memory to A | 1     | 1      |
| XRL A, # data  | Exclusive or immediate to A   | 2     | 2      |
| INC A          | Increment A                   | 1     | 1      |
| DEC A          | Decrement A                   | 1     | 1      |
| CLR A          | Clear A                       | 1     | 1      |
| CPL A          | Complement A                  | 1     | 1      |
| DA A           | Decimal Adjust A              | 1     | 1      |
| SWAP A         | Swap nibbles of A             | 1     | 1      |
| RL A           | Rotate A left                 | 1     | 1      |
| RLC A          | Rotate A left through carry   | 1     | 1      |
| RR A           | Rotate A right                | 1     | 1      |
| RRC A          | Rotate A right through carry  | 1     | 1      |

| Input/Output |                           |       |        |
|--------------|---------------------------|-------|--------|
| Mnemonic     | Description               | Bytes | Cycles |
| IN A, P      | Input port to A           | 1     | 2      |
| OUTL P, A    | Output A to port          | 1     | 2      |
| MOVD A, P    | Input Expander port to A  | 1     | 2      |
| MOVD P, A    | Output A to Expander port | 1     | 2      |
| ANLD P, A    | And A to Expander port    | 1     | 2      |
| ORLD P, A    | Or A to Expander port     | 1     | 2      |

| Registers |                       |       |        |
|-----------|-----------------------|-------|--------|
| Mnemonic  | Description           | Bytes | Cycles |
| INC R     | Increment register    | 1     | 1      |
| INC @R    | Increment data memory | 1     | 1      |

| Branch       |   |       |        |
|--------------|---|-------|--------|
| Mnemonic     | Description                               | Bytes | Cycles |
| JMP addr     | Jump unconditional                        | 2     | 2      |
| JMPP @A      | Jump indirect                             | 1     | 2      |
| DJNZ R, addr | Decrement register and Jump on R not zero | 2     | 2      |

| Branch    |                    |       |        |
|-----------|--------------------|-------|--------|
| Mnemonic  | Description        | Bytes | Cycles |
| JC addr   | Jump on Carry = 1  | 2     | 2      |
| JNC addr  | Jump on Carry = 0  | 2     | 2      |
| JZ addr   | Jump on A Zero     | 2     | 2      |
| JNZ addr  | Jump on A not Zero | 2     | 2      |
| JT1 addr  | Jump on T1 = 1     | 2     | 2      |
| JNT1 addr | Jump on T1 = 0     | 2     | 2      |
| JTF addr  | Jump on timer flag | 2     | 2      |

| Subroutine |                    |       |        |
|------------|--------------------|-------|--------|
| Mnemonic   | Description        | Bytes | Cycles |
| CALL       | Jump to subroutine | 2     | 2      |
| RET        | Return             | 1     | 2      |

| Flags    |                  |       |        |
|----------|------------------|-------|--------|
| Mnemonic | Description      | Bytes | Cycles |
| CLR C    | Clear Carry      | 1     | 1      |
| CPL C    | Complement Carry | 1     | 1      |



Table 2. Instruction Set Summary (cont.)

| Data Moves     |                                   |       |        | Timer/Counter |                    |       |        |
|----------------|-----------------------------------|-------|--------|---------------|--------------------|-------|--------|
| Mnemonic       | Description                       | Bytes | Cycles | Mnemonic      | Description        | Bytes | Cycles |
| MOV A, R       | Move register to A                | 1     | 1      | MOV A, T      | Read Timer/Counter | 1     | 1      |
| MOV A, @R      | Move data memory to A             | 1     | 1      | MOV T, A      | Load Timer/Counter | 1     | 1      |
| MOV A, # data  | Move immediate to A               | 2     | 2      | STRT T        | Start Timer        | 1     | 1      |
| MOV R, A       | Move A to register                | 1     | 1      | STRT CNT      | Start Counter      | 1     | 1      |
| MOV @R, A      | Move A to data memory             | 1     | 1      | STOP TCNT     | Stop Timer/Counter | 1     | 1      |
| MOV R, # data  | Move immediate to register        | 2     | 2      |               |                    |       |        |
| MOV @R, # data | Move immediate to data memory     | 2     | 2      |               |                    |       |        |
| XCH A, R       | Exchange A and register           | 1     | 1      |               |                    |       |        |
| XCH A, @R      | Exchange A and data memory        | 1     | 1      |               |                    |       |        |
| XCHD A, @R     | Exchange nibble of A and register | 1     | 1      |               |                    |       |        |
| MOVP A, @A     | Move to A from current page       | 1     | 2      |               |                    |       |        |

### ABSOLUTE MAXIMUM RATINGS\*

|   |                   |
|---|-------------------|
| Ambient Temperature Under Bias            | 0° C to 70° C     |
| Storage Temperature                       | -65° C to +150° C |
| Voltage on Any Pin with Respect to Ground | -0.5V to +7V      |
| Power Dissipation                         | 1W                |

\*NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

### D.C. CHARACTERISTICS (TA = 0° C to 70° C, VCC = 5.5V ± 1V, VSS = 0V)

| Symbol    | Parameter   | Limits |     |      | Unit | Test Conditions       |
|-----------|---|--------|-----|------|------|-----------------------|
|           |   | Min    | Typ | Max  |      |                       |
| VIL       | Input Low Voltage                                     | -0.5   |     | 0.8  | V    |                       |
| VIH       | Input High Voltage (All except XTAL 1 & 2, T1, RESET) | 3.0    |     | VCC  | V    |                       |
| VIH1      | Input High Voltage (XTAL 1 & 2, T1, RESET)            | 3.8    |     | VCC  | V    |                       |
| VIH(10%)  | Input High Voltage (All except XTAL 1 & 2, T1, RESET) | 2.0    |     | VCC  | V    | VCC = 5.0V ± 10%      |
| VIH1(10%) | Input High Voltage (XTAL 1 & 2, T1, RESET)            | 3.5    |     | VCC  | V    | VCC = 5.0V ± 10%      |
| VOL       | Output Low Voltage                                    |        |     | 0.45 | V    | IOL = 1.6 mA          |
| VOL1      | Output Low Voltage (P10, P11)                         |        |     | 2.5  | V    | IOL = 7 mA            |
| VOH       | Output High Voltage (All unless Open Drain)           | 2.4    |     |      | V    | IOH = 40 µA           |
| ILO       | Output Leakage Current (Open Drain Option—Port 0)     |        |     | ±10  | µA   | VSS + 0.4 ≤ VIN ≤ VCC |
| ICC       | VCC Supply Current                                    |        | 30  | 60   | mA   |                       |

### T1 ZERO CROSS CHARACTERISTICS (TA = 0° C to 70° C, VCC = 5.5V ± 1V, VSS = 0V, CL = 80 pF)

| Symbol | Parameter                                 | Min  | Max  | Unit | Test Conditions             |
|--------|---|------|------|------|-----------------------------|
| VZX    | Zero-Cross Detection Input (T1)           | 1    | 3    | Vpp  | AC Coupled, C = .2µF        |
| AZX    | Zero-Cross Accuracy                       |      | ±135 | mV   | 60 Hz Sine Wave             |
| FZX    | Zero-Cross Detection Input Frequency (T1) | 0.05 | 1    | kHz  |                             |
| tCY    | Cycle Time                                | 8.38 | 50.0 |      | 3.58 MHz XTAL = 8.38 µs tCY |

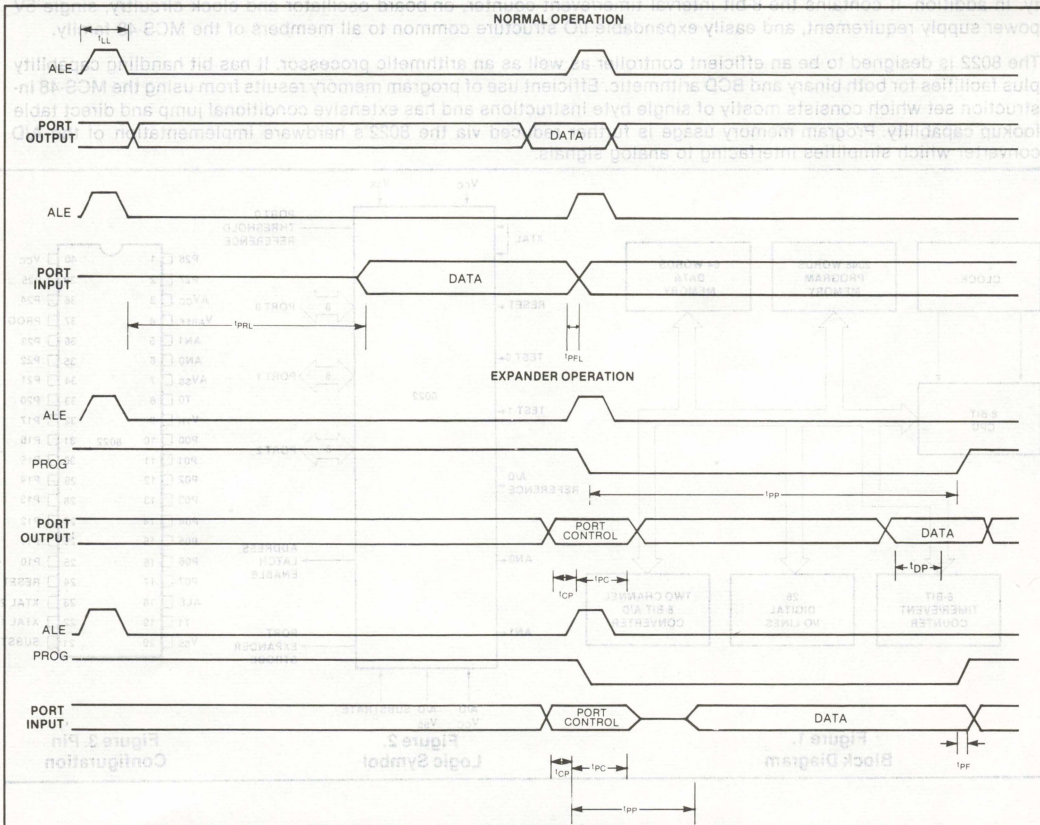


# A.C. CHARACTERISTICS (TA = 0°C to 70°C, VCC = 5.5V ± 1V, VSS = 0V)

Test Conditions: CL = 80 pF, tCY = 8.38 μs

|                    | Symbol | Parameter  | Min. | Max. | Unit | Test Conditions |
|--------------------|--------|--|------|------|------|-----------------|
| Normal Operation   | tCY    | Cycle Time                                       | 8.38 | 50.0 | μs   | 3.58 MHz XTAL   |
|                    | tPRL   | ALE to Time P± Input Must Be Valid (input setup) |      | 6.5  | μs   |                 |
|                    | tPFL   | Input Data Hold Time                             | 0    |      | μs   |                 |
|                    | tLL    | ALE Pulse Width                                  | 0.8  |      | μs   |                 |
|                    | tR     | Reset High                                       | 3    |      | tCY  |                 |
|                    | RXTAL  | Resistor Across XTAL                             | .5   | 1    | MΩ   |                 |
| Expander Operation | tCP    | Port Control Setup Before Falling Edge of PROG   | 0.3  |      | μs   |                 |
|                    | tPC    | Port Control Hold After Falling Edge of PROG     | 0.8  |      | μs   |                 |
|                    | tPR    | PROG to Time P± Input Must Be Valid              | 2.0  | 4.0  | μs   |                 |
|                    | tDP    | Output Data Setup Time                           | 1.0  |      | μs   |                 |
|                    | tPF    | Input Data Hold Time                             | 0    | .15  | μs   |                 |
|                    | tPP    | PROG Pulse Width                                 | 6.0  |      | μs   |                 |

## PORT 2 TIMING





# SINGLE COMPONENT 8-BIT MICROCOMPUTER WITH ON-CHIP A/D CONVERTER

- 8-Bit CPU, ROM, RAM, I/O in Single 40-Pin Package
- On-Chip 8-Bit A/D Converter; Two Input Channels
- 8 Comparator Inputs (Port 0)
- Zero-Cross Detection Capability
- Single 5V Supply (4.5V to 6.5V)
- High Current Drive Capability—2 Pins
- Two Interrupts—External and Timer
- 2K x 8 ROM, 64 x 8 RAM, 28 I/O Lines
- 8.38  $\mu$ sec Cycle; All Instructions 1 or 2 Cycles
- Instructions—8048 Subset
- Interval Timer/Event Counter
- Clock Generated with Single Inductor or Crystal
- Easily Expanded I/O

The Intel 8022 is a member of the MCS®-48 family of single chip 8-bit microcomputers. It is designed to satisfy the requirements of low cost, high volume applications which involve analog signals, capacitive touchpanel keyboards, and/or large ROM space. The 8022 addresses these applications by integrating many new functions on-chip, such as A/D conversion, comparator inputs and zero-cross detection.

The features of the 8022 include 2K bytes of program memory (ROM), 64 bytes of data memory (RAM), 28 I/O lines, an on-chip A/D converter with two input channels, an 8-bit port with comparator inputs for interfacing to low voltage capacitive touchpanels or other non-TTL interfaces, external and timer interrupts, and zero-cross detection capability. In addition, it contains the 8-bit interval timer/event counter, on-board oscillator and clock circuitry, single 5V power supply requirement, and easily expandable I/O structure common to all members of the MCS-48 family.

The 8022 is designed to be an efficient controller as well as an arithmetic processor. It has bit handling capability plus facilities for both binary and BCD arithmetic. Efficient use of program memory results from using the MCS-48 instruction set which consists mostly of single byte instructions and has extensive conditional jump and direct table lookup capability. Program memory usage is further reduced via the 8022's hardware implementation of the A/D converter which simplifies interfacing to analog signals.

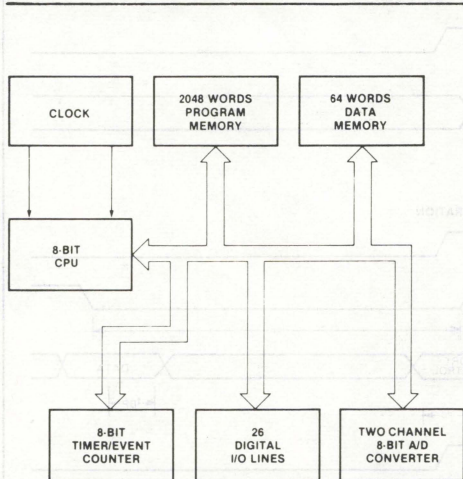


Figure 1.  
Block Diagram

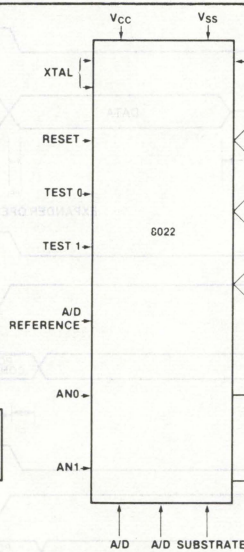


Figure 2.  
Logic Symbol

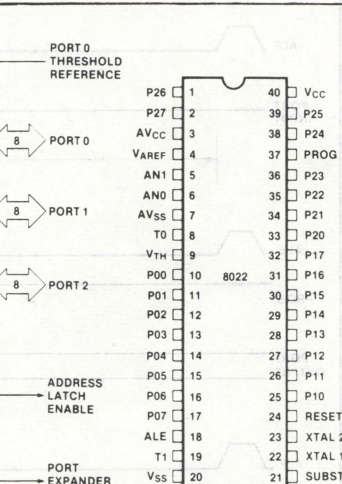


Figure 3. Pin  
Configuration



Table 1. Pin Description

| Designation    | Pin No. | Function  | Designation | Pin No. | Function   |
|----------------|---------|---|-------------|---------|--|
| VSS            | 20      | Circuit GND potential.  | RESET       | 24      | Input used to initialize the processor by clearing status flip-flops and setting the program counter to zero. (Active high).   |
| VCC            | 40      | + 5V circuit power supply.  | AVSS        | 7       | A/D converter GND Potential. Also establishes the lower limit of the conversion range.   |
| PROG           | 37      | Output strobe for Intel® 8243 I/O expander.   | AVCC        | 3       | A/D + 5V power supply.   |
| P00-P07 Port 0 | 10-17   | 8-bit open-drain port with comparator inputs. The switching threshold is set externally by VTH. Optional pull-up resistors may be added via ROM mask selection.   | SUBST       | 21      | Substrate pin used with a bypass capacitor to stabilize the substrate voltage and improve A/D accuracy.                        |
| VTH            | 9       | Port 0 threshold reference pin.   | VAREF       | 4       | A/D converter reference voltage. Establishes the upper limit of the conversion range.  |
| P10-P17 Port 1 | 25-32   | 8-bit quasi-bidirectional port.   | AN0, AN1    | 6,5     | Analog inputs to A/D converter. Software selectable on-chip via SEL AN0 and SEL AN1 instructions.                              |
| P20-P27 Port 2 | 33-36   | 8-bit quasi-bidirectional port.   | ALE         | 18      | Address Latch Enable. Signal occurring once every 30 input clocks (once every cycle), used as an output clock.                 |
| Port 2         | 38-39   | P20-23 also serve as a 4-bit I/O expander for Intel® 8243.  | XTAL 1      | 22      | One side of crystal or inductor input for internal oscillator. Also input for external frequency source. (Not TTL compatible.) |
| T0             | 1-2     | Interrupt input and input pin testable using the conditional transfer instructions JT0 and JNT0. Initiates an interrupt following a low level input if interrupt is enabled. Interrupt is disabled after a reset.   | XTAL 2      | 23      | Other side of timing control element. This pin is not connected when an external frequency source is used.                     |
| T1             | 8       | Input pin testable using the JT1 and JNT1 conditional transfer instructions. Can be designated the timer/event counter input using the STRT CNT instruction. Also serves as the zero-cross detection input to allow zero-crossover sensing of slowly moving AC inputs. Optional pull-up resistor may be added via ROM mask selection. |             |         |  |

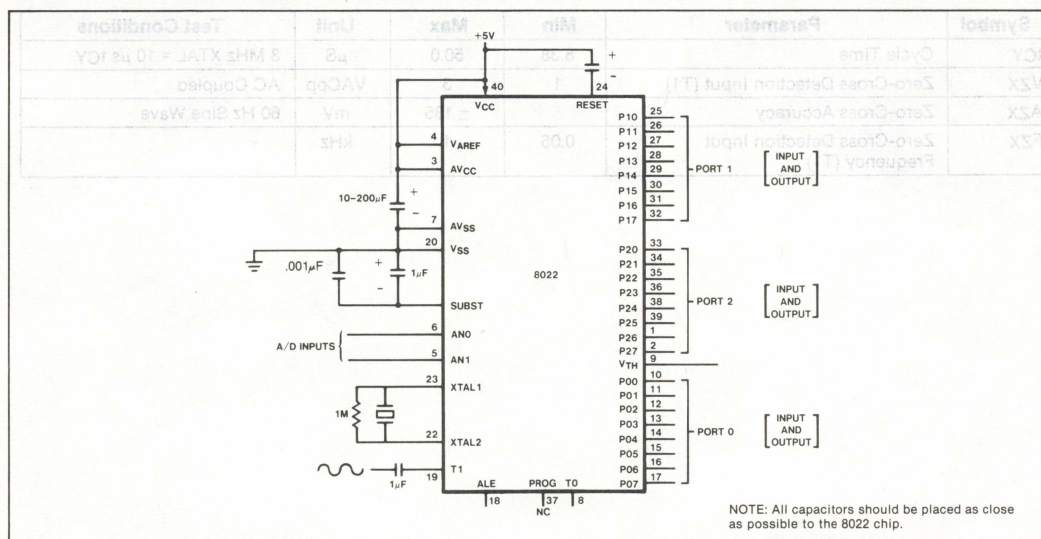


Figure 3. The Stand Alone 8022



# **ABSOLUTE MAXIMUM RATINGS\***

Ambient Temperature Under Bias ..... 0°C to 70°C  
 Storage Temperature ..... -65°C to +150°C  
 Voltage on Any Pin with  
     Respect to Ground ..... -0.5V to +7V  
 Power Dissipation ..... 1 Watt

*\*NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

## **D.C. CHARACTERISTICS** $T_A = 0^\circ\text{C to } 70^\circ\text{C}$ , $V_{CC} = 5.5\text{V} \pm 1\text{V}$ , $V_{SS} = 0\text{V}$

| Symbol    | Parameter  | Limits       |     |              | Unit          | Test Conditions   |
|-----------|--|--------------|-----|--------------|---------------|---|
|           |  | Min          | Typ | Max          |               |   |
| $V_{IL}$  | Input Low Voltage  | -0.5         |     | 0.8          | V             | $V_{TH}$ Floating   |
| $V_{IL1}$ | Input Low Voltage (Port 0)                                   | -0.5         |     | $V_{TH}-0.1$ | V             |   |
| $V_{IH}$  | High Voltage<br>(All except XTAL 1, RESET)                   | 2.0          |     | $V_{CC}$     | V             | $V_{CC} = 5.0\text{V} \pm 10\%$<br>$V_{TH}$ Floating      |
| $V_{IH1}$ | Input High Voltage<br>(All except XTAL 1, RESET)             | 3.0          |     | $V_{CC}$     | V             | $V_{CC} = 5.5\text{V} \pm 1\text{V}$<br>$V_{TH}$ Floating |
| $V_{IH2}$ | Input High Voltage (Port 0)                                  | $V_{TH}+0.1$ |     | $V_{CC}$     | V             |   |
| $V_{IH3}$ | Input High Voltage (RESET, XTAL 1)                           | 3.0          |     | $V_{CC}$     | V             | $V_{CC} = 5.0\text{V} \pm 10\%$                           |
| $V_{TH}$  | Port 0 Threshold Reference Voltage                           | 0            |     | $.4V_{CC}$   | V             |   |
| $V_{OL}$  | Output Low Voltage   |              |     | 0.45         | V             | $I_{OL} = 1.6\text{ mA}$                                  |
| $V_{OL1}$ | Output Low Voltage (P10, P11)                                |              |     | 2.5          | V             | $I_{OL} = 7\text{ mA}$                                    |
| $V_{OH}$  | Output High Voltage (all unless<br>Open Drain Option—Port 0) | 2.4          |     |              | V             | $I_{OH} = 50\text{ }\mu\text{A}$                          |
| $I_{LI}$  | Input Current (T1)   |              |     | $\pm 200$    | $\mu\text{A}$ | $V_{CC} \geq V_{IN} \geq V_{SS} + .45\text{V}$            |
| $I_{LO}$  | Output Leakage Current<br>(Open Drain Option—Port 0)         |              |     | $\pm 10$     | $\mu\text{A}$ | $V_{CC} \geq V_{IN} \geq V_{SS} + 0.45\text{V}$           |
| $I_{CC}$  | $V_{CC}$ Supply Current                                      |              | 50  | 80           | mA            |   |

## **A.C. CHARACTERISTICS** $T_A = 0^\circ\text{C to } 70^\circ\text{C}$ , $V_{CC} = 5.5\text{V} \pm 1\text{V}$ , $V_{SS} = 0\text{V}$

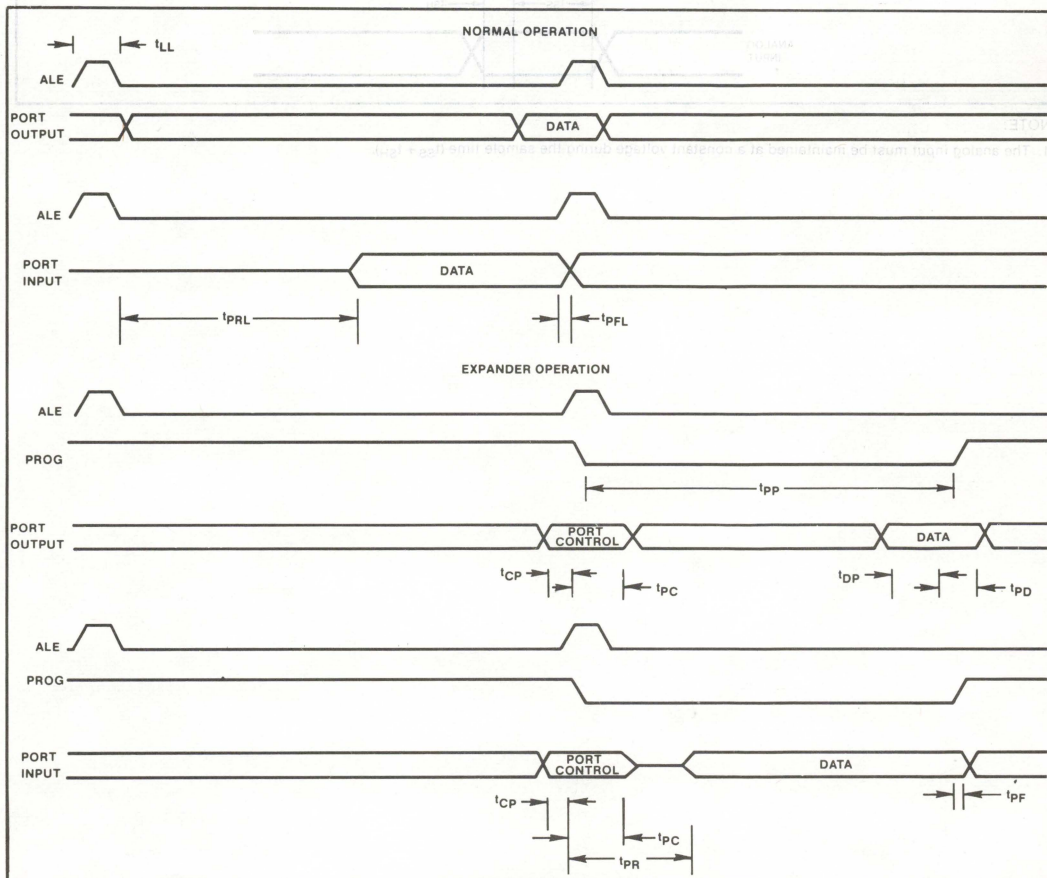
| Symbol   | Parameter                                    | Min  | Max       | Unit          | Test Conditions                        |
|----------|--|------|-----------|---------------|--|
| $t_{CY}$ | Cycle Time                                   | 8.38 | 50.0      | $\mu\text{s}$ | 3 MHz XTAL = 10 $\mu\text{s}$ $t_{CY}$ |
| $V_{ZX}$ | Zero-Cross Detection Input (T1)              | 1    | 3         | VACpp         | AC Coupled                             |
| $A_{ZX}$ | Zero-Cross Accuracy                          |      | $\pm 135$ | mV            | 60 Hz Sine Wave                        |
| $F_{ZX}$ | Zero-Cross Detection Input<br>Frequency (T1) | 0.05 | 1         | kHz           |  |

**A.C. CHARACTERISTICS**  $T_A = 0^\circ\text{C to } 70^\circ\text{C}$ ,  $V_{CC} = 5.5\text{V} \pm 1\text{V}$ ,  $V_{SS} = 0\text{V}$

Test Conditions:  $C_L = 80\text{ pF}$   $t_{CY} = 8.38\text{ }\mu\text{s}$

|                    | Symbol    | Parameter                                      | Min | Max  | Unit          | Notes                                      |
|--------------------|-----------|--|-----|------|---------------|--|
|                    | $t_{CP}$  | Port Control Setup Before Falling Edge of PROG | 0.5 |      | $\mu\text{s}$ |  |
|                    | $t_{PC}$  | Port Control Hold After Falling Edge of PROG   | 0.8 |      | $\mu\text{s}$ |  |
| Expander Operation | $t_{PR}$  | PROG to Time P2 Input Must Be Valid            |     | 1.0  | $\mu\text{s}$ |  |
|                    | $t_{DP}$  | Output Data Setup Time                         | 7.0 |      | $\mu\text{s}$ |  |
|                    | $t_{PD}$  | Output Data Hold Time                          | 8.3 |      | $\mu\text{s}$ |  |
|                    | $t_{PF}$  | Input Data Hold Time                           | 0   | .150 | $\mu\text{s}$ |  |
|                    | $t_{PP}$  | PROG Pulse Width                               | 8.3 |      | $\mu\text{s}$ |  |
|                    | $t_{PRL}$ | ALE to Time P2 Input Must Be Valid             |     | 3.6  | $\mu\text{s}$ |  |
|                    | $t_{PFL}$ | Input Data Hold Time                           | 0   |      | $\mu\text{s}$ |  |
|                    | $t_{LL}$  | ALE Pulse Width                                | 3.9 | 23.0 | $\mu\text{s}$ | $t_{CY} = 8.38\text{ }\mu\text{s}$ for min |

**Port 2 Timing**

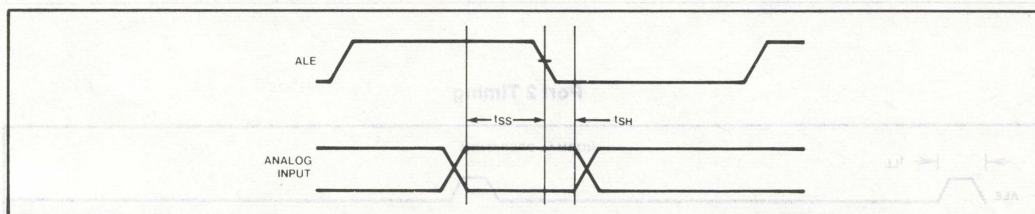




**A/D CONVERTER CHARACTERISTICS**  $T_A = 0^\circ\text{C to } 70^\circ\text{C}$ ,  $V_{CC} = 5.5\text{V} \pm 1\text{V}$ ,  $V_{SS} = 0\text{V}$ ,  $AV_{CC} = 5.5\text{V} \pm 1\text{V}$ ,  $AV_{SS} = 0\text{V}$ ,  $AV_{CC}/2 \leq V_{AREF} \leq AV_{CC}$

| Parameter  | Min | Typ  | Max                   | Unit     | Comments |
|--|-----|------|-----------------------|----------|----------|
| Resolution   | 8   |      |                       | Bits     |          |
| Absolute Accuracy                                    |     |      | .8% FSR $\pm$ 1/2 LSB | LSB      | (Note 1) |
| Sample Setup Before Falling Edge of ALE ( $t_{SS}$ ) |     | 0.20 |                       | $t_{CY}$ |          |
| Sample Hold After Falling Edge of ALE ( $t_{SH}$ )   |     | 0.10 |                       | $t_{CY}$ |          |
| Input Capacitance (AN0, AN1)                         |     | 1    |                       | pF       |          |
| Conversion Time                                      | 4   |      | 4                     | $t_{CY}$ |          |

**Analog Input Timing**



**NOTE:**

1. The analog input must be maintained at a constant voltage during the sample time ( $t_{SS} + t_{SH}$ ).

Table 2. Instruction Set Summary

|              | Mnemonic                | Description                               | Bytes | Cycle | Hexadecimal Opcode      |
|--------------|-------------------------|---|-------|-------|-------------------------|
| Accumulator  | ADD A,R <sub>r</sub>    | Add register to A                         | 1     | 1     | 68-6F                   |
|              | ADD A,@R                | Add data memory to A                      | 1     | 1     | 60-61                   |
|              | ADD A,#data             | Add immediate to A                        | 2     | 2     | 03                      |
|              | ADDC A,R <sub>r</sub>   | Add register with carry                   | 1     | 1     | 78-7F                   |
|              | ADDC A,@R               | Add data memory with carry                | 1     | 1     | 70-71                   |
|              | ADDC A,#data            | Add immediate with carry                  | 2     | 2     | 13                      |
|              | ANL A,R <sub>r</sub>    | And register to A                         | 1     | 1     | 58-5F                   |
|              | ANL A,@R                | And data memory to A                      | 1     | 1     | 50-51                   |
|              | ANL A,#data             | And immediate to A                        | 2     | 2     | 53                      |
|              | ORL A,R <sub>r</sub>    | Or register to A                          | 1     | 1     | 48-4F                   |
|              | ORL A,@R                | Or data memory to A                       | 1     | 1     | 40-41                   |
|              | ORL A,#data             | Or immediate to A                         | 2     | 2     | 43                      |
|              | XRL A,R <sub>r</sub>    | Exclusive Or register to A                | 1     | 1     | D8-DF                   |
|              | XRL A,@R                | Exclusive Or data memory to A             | 1     | 1     | D0-D1                   |
|              | XRL A,#data             | Exclusive Or immediate to A               | 2     | 2     | D3                      |
|              | INC A                   | Increment A                               | 1     | 1     | 17                      |
|              | DEC A                   | Decrement A                               | 1     | 1     | 07                      |
|              | CLR A                   | Clear A                                   | 1     | 1     | 27                      |
|              | CPL A                   | Complement A                              | 1     | 1     | 37                      |
|              | DA A                    | Decimal adjust A                          | 1     | 1     | 57                      |
| Input/Output | SWAP A                  | Swap nibbles of A                         | 1     | 1     | 47                      |
|              | RL A                    | Rotate A left                             | 1     | 1     | E7                      |
|              | RLC A                   | Rotate A left through carry               | 1     | 1     | F7                      |
|              | RR A                    | Rotate A right                            | 1     | 1     | 77                      |
| Registers    | RRC A                   | Rotate A right through carry              | 1     | 1     | 67                      |
|              | IN A,P <sub>p</sub>     | Input port to A                           | 1     | 2     | 08,09,0A                |
|              | OUTL P <sub>p</sub> ,A  | Output A to port                          | 1     | 2     | 90,99,3A                |
|              | MOV D,P <sub>p</sub> ,A | Input expander port to A                  | 1     | 2     | 0C-0F                   |
| Branch       | MOV D,P <sub>p</sub> ,A | Output A to expander port                 | 1     | 2     | 3C-3F                   |
|              | ANLD P <sub>p</sub> ,A  | And A to expander port                    | 1     | 2     | 9C-9F                   |
|              | ORLD P <sub>p</sub> ,A  | Or A to expander port                     | 1     | 2     | 8C-8F                   |
|              | INC R <sub>r</sub>      | Increment register                        | 1     | 1     | 18-1F                   |
|              | INC @R                  | Increment data memory                     | 1     | 1     | 10-11                   |
|              | JMP addr                | Jump unconditional                        | 2     | 2     | 04,24,44,64,84,A4,C4,E4 |
|              | JMPP @A                 | Jump indirect                             | 1     | 2     | B3                      |
|              | DJNZ R,addr             | Decrement register and jump on R not zero | 2     | 2     | E8-EF                   |
|              | JC addr                 | Jump on carry=1                           | 2     | 2     | F6                      |
|              | JNC addr                | Jump on carry=0                           | 2     | 2     | E6                      |
|              | JZ addr                 | Jump on A zero                            | 2     | 2     | C6                      |
|              | JNZ addr                | Jump on A not zero                        | 2     | 2     | 96                      |
|              |                         |   |       |       |                         |
|              |                         |   |       |       |                         |

|               | Mnemonic                  | Description                          | Bytes | Cycle | Hexadecimal Opcode      |
|---------------|---------------------------|--------------------------------------|-------|-------|-------------------------|
|               | JTO addr                  | Jump on T0=1                         | 2     | 2     | 36                      |
|               | JNTO addr                 | Jump on T0=0                         | 2     | 2     | 26                      |
|               | JT1 addr                  | Jump on T1=1                         | 2     | 2     | 56                      |
|               | JNT1 addr                 | Jump on T1=0                         | 2     | 2     | 46                      |
|               | JTF addr                  | Jump on timer flag                   | 2     | 2     | 16                      |
|               |                           |                                      |       |       |                         |
|               |                           |                                      |       |       |                         |
|               |                           |                                      |       |       |                         |
| Subroutine    | CALL addr                 | Jump to subroutine                   | 1     | 2     | 14,34,54,74,94,B4,D4,F4 |
|               | RET                       | Return                               | 1     | 2     | 83                      |
| Flags         | CLR C                     | Clear carry                          | 1     | 1     | 97                      |
|               | CPL C                     | Complement carry                     | 1     | 1     | A7                      |
| Data Moves    | MOV A,R <sub>r</sub>      | Move register to A                   | 1     | 1     | F8-FF                   |
|               | MOV A,@R                  | Move data memory to A                | 1     | 1     | F0-F1                   |
|               | MOV A,#data               | Move immediate to A                  | 2     | 2     | 23                      |
|               | MOV R <sub>r</sub> ,A     | Move A to register                   | 1     | 1     | A8-AF                   |
|               | MOV @R,A                  | Move A to data memory                | 1     | 1     | A0-A1                   |
|               | MOV R <sub>r</sub> ,#data | Move immediate to register           | 2     | 2     | B8-BF                   |
|               | MOV @R,#data              | Move immediate to data memory        | 2     | 2     | B0-B1                   |
|               | XCH A,R <sub>r</sub>      | Exchange A and register              | 1     | 1     | 28-2F                   |
|               | XCH A,@R                  | Exchange A and data memory           | 1     | 1     | 20-21                   |
|               | XCHD a,@R                 | Exchange nibble of A and register    | 1     | 1     | 30-31                   |
| Timer/Counter | MOVP A,@A                 | Move to A from current page          | 1     | 2     | A3                      |
|               | MOV A,T                   | Read timer/counter                   | 1     | 1     | 42                      |
|               | MOV T,A                   | Load timer/counter                   | 1     | 1     | 62                      |
|               | STRT T                    | Start timer                          | 1     | 1     | 55                      |
| A/D Converter | STRT CNT                  | Start counter                        | 1     | 1     | 45                      |
|               | STOP TCNT                 | Stop timer/counter                   | 1     | 1     | 65                      |
|               | RAD                       | Move conversion result register to A | 1     | 2     | 80                      |
|               | SEL AN0                   | Select analog input zero             | 1     | 1     | 85                      |
| Interrupts    | SEL AN1                   | Select analog input one              | 1     | 1     | 95                      |
|               | EN I                      | Enable external interrupt            | 1     | 1     | 05                      |
|               | DIS I                     | Disable external interrupt           | 1     | 1     | 15                      |
|               | EN TCNTI                  | Enable timer/counter interrupt       | 1     | 1     | 25                      |
|               | DIS TCNTI                 | Disable timer/counter interrupt      | 1     | 1     | 35                      |
|               | RET I                     | Return from interrupt                | 1     | 2     | 93                      |
|               | NOP                       | No operation                         | 1     | 1     | 00                      |

SYMBOLS AND ABBREVIATIONS USED

A Accumulator  
 addr 11-Bit Program Memory Address  
 ANO, AN1 Analog Input 0, Analog Input 1  
 CNT Event Counter  
 data 8-Bit Number or Expression  
 I Interrupt

P Mnemonic for "in-page" Operation  
 P<sub>p</sub> Port Designator (P=0, 1, 2 or 4-7)  
 R<sub>r</sub> Register Designator (r=0-7)  
 T Timer  
 T0, T1 Test 0, Test 1  
 # Immediate Data Prefix  
 @ Indirect Address Prefix



- High Performance HMOS
- Interval Timer/Event Counter
- Two Single Level Interrupts
- Single 5-Volt Supply
- Over 96 Instructions; 90% Single Byte

- Reduced Power Consumption
- Compatible with 8080/8085 Peripherals
- Easily Expandable Memory and I/O
- Up to 1.36  $\mu$  sec Instruction Cycle
- All Instructions 1 or 2 cycles

The Intel MCS<sup>®</sup>-48 family are totally self-sufficient, 8-bit parallel computers fabricated on single silicon chips using Intel's advanced N-channel silicon gate HMOS process.

The family contains 27 I/O lines, an 8-bit timer/counter, and on-board oscillator/clock circuits. For systems that require extra capability, the family can be expanded using MCS<sup>®</sup>-80/MCS<sup>®</sup>-85 peripherals.

To minimize development problems and provide maximum flexibility, a logically and functionally pin-compatible version of the ROM devices with UV-erasable user-programmable EPROM program memory is available with minor differences.

These microcomputers are designed to be efficient controllers as well as arithmetic processors. They have extensive bit handling capability as well as facilities for both binary and BCD arithmetic. Efficient use of program memory results from an instruction set consisting mostly of single byte instructions and no instructions over 2 bytes in length.

| Device | Internal     | Memory      | RAM Standby |
|--------|--------------|-------------|-------------|
| 8050AH | 4K x 8 ROM   | 256 x 8 RAM | yes         |
| 8049H  | 2K x 8 ROM   | 128 x 8 RAM | yes         |
| 8048H  | 1K x 8 ROM   | 64 x 8 RAM  | yes         |
| 8040HL | none         | 256 x 8 RAM | yes         |
| 8039HL | none         | 128 x 8 RAM | yes         |
| 8035HL | none         | 64 x 8 RAM  | yes         |
| 8749H  | 2K x 8 EPROM | 128 x 8 RAM | no          |
| 8748H  | 1K x 8 EPROM | 64 x 8 RAM  | no          |

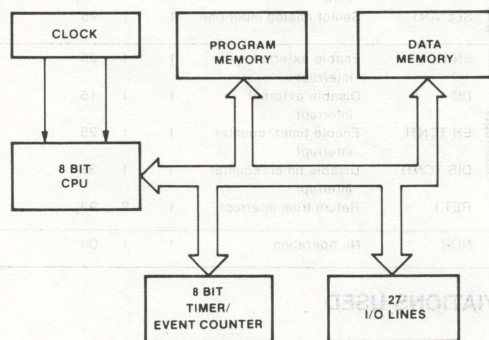


Figure 1.  
Block Diagram

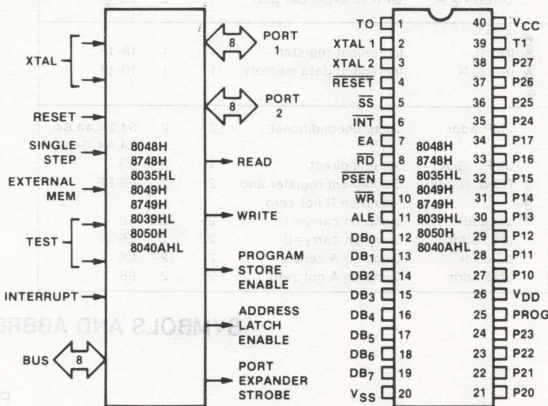


Figure 2.  
Logic Symbol

Figure 3.  
Pin Configuration



Table 1. Pin Description

| Symbol         | Pin No.        | Function  | Device              | Symbol | Pin No. | Function   | Device  |
|----------------|----------------|---|---------------------|--------|---------|--|---|
| VSS            | 20             | Circuit GND potential   | All                 | RD     | 8       | Output strobe activated during a BUS read. Can be used to enable data onto the bus from an external device.<br>Used as a read strobe to external data memory. (Active low)                         | All   |
| VDD            | 26             | +5V during normal operation.<br>Low power standby pin.  | All                 | RESET  | 4       | Input which is used to initialize the processor. (Active low) (Non TTL V <sub>IH</sub> )<br>Used during power down.<br>Used during programming.<br>Used during ROM verification.                   | All<br>8048H, 8035HL, 8049H, 8039HL, 8050AH, 8040AHL<br>8748H, 8749H<br>8048H, 8748H, 8049H, 8749H, 8050AH. |
| VCC            | 40             | Main power supply; +5V during operation and programming.  | All                 | WR     | 10      | Output strobe during a bus write. (Active low)<br>Used as write strobe to external data memory.  | All   |
| PROG           | 25             | Output strobe for 8243 I/O expander.<br>Program pulse (+18V) input pin during programming.  | All<br>8748H, 8749H | ALE    | 11      | Address latch enable. This signal occurs once during each cycle and is useful as a clock output.<br>The negative edge of ALE strobes address into external data and program memory.                | All   |
| P10-P17 Port 1 | 27-34          | 8-bit quasi-bidirectional port.   | All                 | PSEN   | 9       | Program store enable. This output occurs only during a fetch to external program memory. (Active low)  | All   |
| P20-P27 Port 2 | 21-24<br>35-38 | 8-bit quasi-bidirectional port. P20-P23 contain the four high order program counter bits during an external program memory fetch and serve as a 4-bit I/O expander bus for 8243.  | All                 | SS     | 5       | Single step input can be used in conjunction with ALE to "single step" the processor through each instruction. (Active low)  | All   |
| DB0-DB7 BUS    | 12-19          | True bidirectional port which can be written or read synchronously using the RD, WR strobes. The port can also be statically latched.<br>Contains the 8 low order program counter bits during an external program memory fetch, and receives the addressed instruction under the control of PSEN. Also contains the address and data during an external RAM data store instruction, under control of ALE, RD, and WR. | All                 | EA     | 7       | External access input which forces all program memory fetches to reference external memory. Useful for emulation and debug.<br>Used during (18V) programming<br>Used during ROM verification (12V) | All<br>8748H, 8749H<br>8048H, 8049H, 8050AH   |
| T0             | 1              | Input pin testable using the conditional transfer instructions JT0 and JNT0. T0 can be designated as a clock output using ENT0 CLK instruction.<br>Used during programming.   | All<br>8748H, 8749H | XTAL1  | 2       | One side of crystal input for internal oscillator. Also input for external source. (Non TTL V <sub>IH</sub> )  | All   |
| T1             | 39             | Input pin testable using the JT1, and JNT1 instructions. Can be designated the timer/counter input using the STRT CNT instruction.  | All                 | XTAL2  | 3       | Other side of crystal input.   | All   |
| INT            | 6              | Interrupt input. Initiates an interrupt if interrupt is enabled. Interrupt is disabled after a reset. Also testable with conditional jump instruction. (Active low) Interrupt must remain low for at least 3 machine cycles for proper operation.   | All                 |        |         |  |   |



Table 2. Instruction Set

| Accumulator    |                               |       |        |
|----------------|-------------------------------|-------|--------|
| Mnemonic       | Description                   | Bytes | Cycles |
| ADD A, R       | Add register to A             | 1     | 1      |
| ADD A, @R      | Add data memory to A          | 1     | 1      |
| ADD A, # data  | Add immediate to A            | 2     | 2      |
| ADDC A, R      | Add register with carry       | 1     | 1      |
| ADDC A, @R     | Add data memory with carry    | 1     | 1      |
| ADDC A, # data | Add immediate with carry      | 2     | 2      |
| ANL A, R       | And register to A             | 1     | 1      |
| ANL A, @R      | And data memory to A          | 1     | 1      |
| ANL A, # data  | And immediate to A            | 2     | 2      |
| ORL A, R       | Or register to A              | 1     | 1      |
| ORL A, @R      | Or data memory to A           | 1     | 1      |
| ORL A, # data  | Or immediate to A             | 2     | 2      |
| XRL A, R       | Exclusive or register to A    | 1     | 1      |
| XRL A, @R      | Exclusive or data memory to A | 1     | 1      |
| XRL A, # data  | Exclusive or immediate to A   | 2     | 2      |
| INC A          | Increment A                   | 1     | 1      |
| DEC A          | Decrement A                   | 1     | 1      |
| CLR A          | Clear A                       | 1     | 1      |
| CPL A          | Complement A                  | 1     | 1      |
| DA A           | Decimal adjust A              | 1     | 1      |
| SWAP A         | Swap nibbles of A             | 1     | 1      |
| RL A           | Rotate A left                 | 1     | 1      |
| RLC A          | Rotate A left through carry   | 1     | 1      |
| RR A           | Rotate A right                | 1     | 1      |
| RRC A          | Rotate A right through carry  | 1     | 1      |

| Input/Output    |                           |       |        |
|-----------------|---------------------------|-------|--------|
| Mnemonic        | Description               | Bytes | Cycles |
| IN A, P         | Input port to A           | 1     | 2      |
| OUTL P, A       | Output A to port          | 1     | 2      |
| ANL P, # data   | And immediate to port     | 2     | 2      |
| ORL P, # data   | Or immediate to port      | 2     | 2      |
| INS A, BUS      | Input BUS to A            | 1     | 2      |
| OUTL BUS, A     | Output A to BUS           | 1     | 2      |
| ANL BUS, # data | And immediate to BUS      | 2     | 2      |
| ORL BUS, # data | Or immediate to BUS       | 2     | 2      |
| MOVD A, P       | Input expander port to A  | 1     | 2      |
| MOVD P, A       | Output A to expander port | 1     | 2      |
| ANLD P, A       | And A to expander port    | 1     | 2      |
| ORLD P, A       | Or A to expander port     | 1     | 2      |

| Registers |                       |       |        |
|-----------|-----------------------|-------|--------|
| Mnemonic  | Description           | Bytes | Cycles |
| INC R     | Increment register    | 1     | 1      |
| INC @R    | Increment data memory | 1     | 1      |
| DEC R     | Decrement register    | 1     | 1      |

| Branch       |                             |       |        |
|--------------|-----------------------------|-------|--------|
| Mnemonic     | Description                 | Bytes | Cycles |
| JMP addr     | Jump unconditional          | 2     | 2      |
| JMPP @A      | Jump indirect               | 1     | 2      |
| DJNZ R, addr | Decrement register and skip | 2     | 2      |
| JC addr      | Jump on carry = 1           | 2     | 2      |
| JNC addr     | Jump on carry = 0           | 2     | 2      |
| JZ addr      | Jump on A zero              | 2     | 2      |
| JNZ addr     | Jump on A not zero          | 2     | 2      |
| JT0 addr     | Jump on T0 = 1              | 2     | 2      |
| JNT0 addr    | Jump on T0 = 0              | 2     | 2      |
| JT1 addr     | Jump on T1 = 1              | 2     | 2      |
| JNT1 addr    | Jump on T1 = 0              | 2     | 2      |
| JF0 addr     | Jump on F0 = 1              | 2     | 2      |
| JF1 addr     | Jump on F1 = 1              | 2     | 2      |
| JTF addr     | Jump on timer flag          | 2     | 2      |
| JNI addr     | Jump on INT = 0             | 2     | 2      |
| JBb addr     | Jump on accumulator bit     | 2     | 2      |

| Subroutine |                           |       |        |
|------------|---------------------------|-------|--------|
| Mnemonic   | Description               | Bytes | Cycles |
| CALL addr  | Jump to subroutine        | 2     | 2      |
| RET        | Return                    | 1     | 2      |
| RETR       | Return and restore status | 1     | 2      |

| Flags    |                   |       |        |
|----------|-------------------|-------|--------|
| Mnemonic | Description       | Bytes | Cycles |
| CLR C    | Clear carry       | 1     | 1      |
| CPL C    | Complement carry  | 1     | 1      |
| CLR F0   | Clear flag 0      | 1     | 1      |
| CPL F0   | Complement flag 0 | 1     | 1      |
| CLR F1   | Clear flag 1      | 1     | 1      |
| CPL F1   | Complement flag 1 | 1     | 1      |

| Data Moves    |                                   |       |        |
|---------------|-----------------------------------|-------|--------|
| Mnemonic      | Description                       | Bytes | Cycles |
| MOV A, R      | Move register to A                | 1     | 1      |
| MOV A, @R     | Move data memory to A             | 1     | 1      |
| MOV A, # data | Move immediate to A               | 2     | 2      |
| MOV R, A      | Move A to register                | 1     | 1      |
| MOV @R, A     | Move A to data memory             | 1     | 1      |
| MOV-R, # data | Move immediate to register        | 2     | 2      |
| MOV @R, #data | Move immediate to data memory     | 2     | 2      |
| MOV A, PSW    | Move PSW to A                     | 1     | 1      |
| MOV PSW, A    | Move A to PSW                     | 1     | 1      |
| XCH A, R      | Exchange A and register           | 1     | 1      |
| XCH A, @R     | Exchange A and data memory        | 1     | 1      |
| XCHD A, @R    | Exchange nibble of A and register | 1     | 1      |
| MOVX A, @R    | Move external data memory to A    | 1     | 2      |
| MOVX @R, A    | Move A to external data memory    | 1     | 2      |
| MOVP A, @A    | Move to A from current page       | 1     | 2      |
| MOVVP3 A, @   | Move to A from page 3             | 1     | 2      |

| Timer/Counter |                                 |       |        |
|---------------|---------------------------------|-------|--------|
| Mnemonic      | Description                     | Bytes | Cycles |
| MOV A, T      | Read timer/counter              | 1     | 1      |
| MOV T, A      | Load timer/counter              | 1     | 1      |
| STRT T        | Start timer                     | 1     | 1      |
| STRT CNT      | Start counter                   | 1     | 1      |
| STOP TCNT     | Stop timer/counter              | 1     | 1      |
| EN TCNTI      | Enable timer/counter interrupt  | 1     | 1      |
| DIS TCNTI     | Disable timer/counter interrupt | 1     | 1      |

| Control  |                            |       |        |
|----------|----------------------------|-------|--------|
| Mnemonic | Description                | Bytes | Cycles |
| EN I     | Enable external interrupt  | 1     | 1      |
| DIS I    | Disable external interrupt | 1     | 1      |
| SEL RB0  | Select register bank 0     | 1     | 1      |
| SEL RB1  | Select register bank 1     | 1     | 1      |
| SEL MB0  | Select memory bank 0       | 1     | 1      |
| SEL MB1  | Select memory bank 1       | 1     | 1      |
| ENT0 CLK | Enable clock output on T0  | 1     | 1      |

| Mnemonic | Description  | Bytes | Cycles |
|----------|--------------|-------|--------|
| NOP      | No operation | 1     | 1      |



# **ABSOLUTE MAXIMUM RATINGS\***

Ambient Temperature Under Bias . . . . . 0°C to 70°C  
Storage Temperature . . . . . -65°C to +150°C  
Voltage On Any Pin With Respect  
to Ground . . . . . -0.5V to +7V  
Package Power Dissipation . . . . . 1.5 Watt

\*NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of device at these or any other conditions above those indicated in the operational sections of this specification is not implied.

## **D.C. CHARACTERISTICS** (TA = 0°C to 70°C; VCC = VDD = 5V ± 10%; VSS = 0V)

| Symbol    | Parameter   | Limits |     |      | Unit | Test Conditions              | Device          |
|-----------|---|--------|-----|------|------|------------------------------|-----------------|
|           |   | Min    | Typ | Max  |      |                              |                 |
| VIL       | Input Low Voltage<br>(All except RESET, X1, X2)               | — .5   |     | .8   | V    |                              | All             |
| VIL1      | Input Low Voltage<br>(RESET, X1, X2)                          | — .5   |     | .6   | V    |                              | All             |
| VIH       | Input High Voltage<br>(All Except XTAL1, XTAL2,<br>RESET)     | 2.0    |     | VCC  | V    |                              | All             |
| VIH1      | Input High Voltage<br>(X1, X2, RESET)                         | 3.8    |     | VCC  | V    |                              | All             |
| VOL       | Output Low Voltage (BUS)                                      |        |     | .45  | V    | IOL = 2.0 mA                 | All             |
| VOL1      | Output Low Voltage<br>(RD, WR, PSEN, ALE)                     |        |     | .45  | V    | IOL = 1.8 mA                 | All             |
| VOL2      | Output Low Voltage (PROG)                                     |        |     | .45  | V    | IOL = 1.0 mA                 | All             |
| VOL3      | Output Low Voltage<br>(All Other Outputs)                     |        |     | .45  | V    | IOL = 1.6 mA                 | All             |
| VOH       | Output High Voltage (BUS)                                     | 2.4    |     |      | V    | IOH = —400 µA                | All             |
| VOH1      | Output High Voltage<br>(RD, WR, PSEN, ALE)                    | 2.4    |     |      | V    | IOH = —100 µA                | All             |
| VOH2      | Output High Voltage<br>(All Other Outputs)                    | 2.4    |     |      | V    | IOH = —40 µA                 | All             |
| IL1       | Input Leakage Current<br>(T1, INT)                            |        |     | ± 10 | µA   | VSS ≤ VIN ≤ VCC              | All             |
| IL11      | Input Leakage Current<br>(P10-P17, P20-P27, EA, SS)           |        |     | —500 | µA   | VSS + .45 ≤ VIN ≤ VCC        | All             |
| IL0       | Output Leakage Current<br>(BUS, T0)<br>(High Impedance State) |        |     | ± 10 | µA   | VSS + .45 ≤ VIN ≤ VCC        | All             |
| IDD       | VDD Supply Current  |        | 4   | 8    | mA   |                              | 8048H, 8035HL   |
|           |   |        | 5   | 10   | mA   |                              | 8049H, 8039HL   |
|           |   |        | 10  | 20   | mA   |                              | 8050AH, 8040AHL |
| IDD + ICC | Total Supply Current  |        | 40  | 80   | mA   |                              | 8048H, 8035HL   |
|           |   |        | 50  | 100  | mA   |                              | 8049H, 8039HL   |
|           |   |        | 40  | 80   | mA   |                              | 8050AH, 8040AHL |
|           |   |        | 30  | 90   | mA   |                              | 8748H           |
|           |   |        | 50  | 110  | mA   |                              | 8749H           |
| VDD       | RAM Standby Voltage   | 2.2    |     | 5.5  | V    | Standby Mode Reset<br>≤ VIL1 | 8048H, 8035HL   |
|           |   | 4.5    |     | 5.5  | V    |                              | 8049H, 8039HL   |
|           |   | 2.2    |     | 5.5  | V    |                              | 8050AH, 8040AHL |



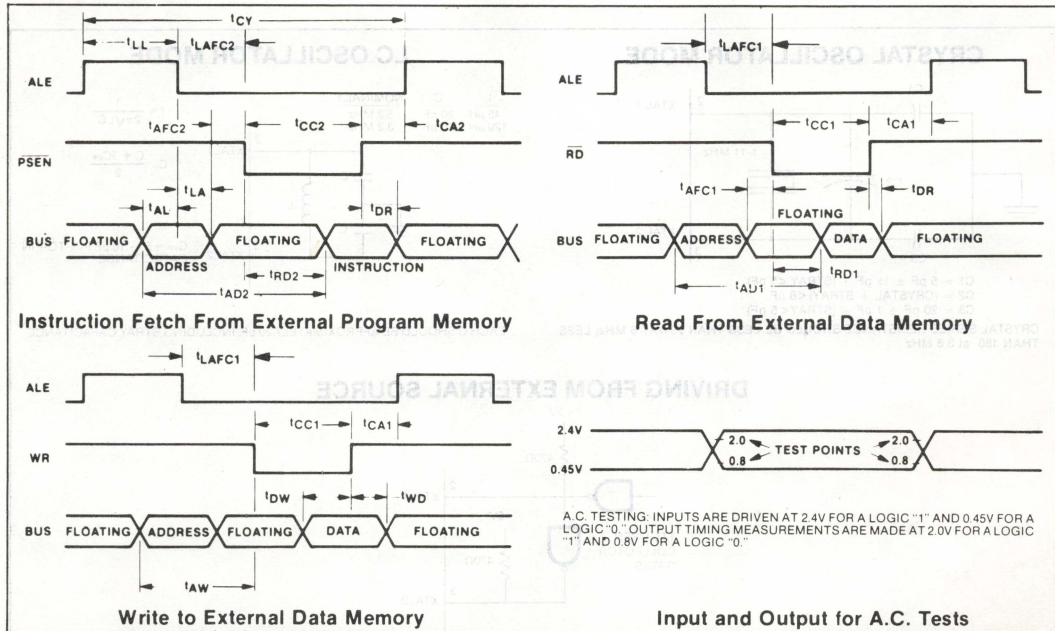
**A.C. CHARACTERISTICS** ( $T_A = 0^\circ\text{C to } 70^\circ\text{C}$ ;  $V_{CC} = V_{DD} = 5V \pm 10\%$ ;  $V_{SS} = 0V$ )

| Symbol             | Parameter  | f (t <sub>CY</sub> )<br>(Note 4) | (Note 3)<br>11 MHz |      | Unit | Conditions<br>(Note 1) |
|--------------------|--|----------------------------------|--------------------|------|------|------------------------|
|                    |  |                                  | Min                | Max  |      |                        |
| t <sub>CY</sub>    | Cycle Time   | 15/F(XTAL)                       | 1.36               | 15.0 | us   | (Note 4)               |
| t <sub>LL</sub>    | ALE Pulse Width  | 7/30 t <sub>CY</sub> -170        | 150                |      | ns   |                        |
| t <sub>AL</sub>    | Addr Setup to ALE  | 2/15 t <sub>CY</sub> -110        | 70                 |      | ns   | (Note 2)               |
| t <sub>LA</sub>    | Addr Hold from ALE   | 1/15 t <sub>CY</sub> -40         | 50                 |      | ns   |                        |
| t <sub>CC1</sub>   | Control Pulse Width ( $\overline{RD}$ , $\overline{WR}$ )                | 1/2 t <sub>CY</sub> -200         | 480                |      | ns   |                        |
| t <sub>CC2</sub>   | Control Pulse Width ( $\overline{PSEN}$ )                                | 2/5 t <sub>CY</sub> -200         | 350                |      | ns   |                        |
| t <sub>DW</sub>    | Data Setup before $\overline{WR}$  | 13/30 t <sub>CY</sub> -200       | 390                |      | ns   |                        |
| t <sub>WD</sub>    | Data Hold after $\overline{WR}$  | 1/15 t <sub>CY</sub> -50         | 40                 |      | ns   |                        |
| t <sub>DR</sub>    | Data Hold ( $\overline{RD}$ , $\overline{PSEN}$ )                        | 1/10 t <sub>CY</sub> -30         | 0                  | 110  | ns   |                        |
| t <sub>RD1</sub>   | $\overline{RD}$ to Data in   | 11/30 t <sub>CY</sub> -170       |                    | 330  | ns   |                        |
| t <sub>RD2</sub>   | $\overline{PSEN}$ to Data in   | 4/15 t <sub>CY</sub> -170        |                    | 190  | ns   |                        |
| t <sub>AW</sub>    | Addr Setup to $\overline{WR}$  | 1/3 t <sub>CY</sub> -150         | 300                |      | ns   |                        |
| t <sub>AD1</sub>   | Addr Setup to Data ( $\overline{RD}$ )                                   | 7/10 t <sub>CY</sub> -220        |                    | 730  | ns   |                        |
| t <sub>AD2</sub>   | Addr Setup to Data ( $\overline{PSEN}$ )                                 | 1/2 t <sub>CY</sub> -220         |                    | 460  | ns   |                        |
| t <sub>AFC1</sub>  | Addr Float to $\overline{RD}$ , $\overline{WR}$                          | 2/15 t <sub>CY</sub> -40         | 140                |      | ns   | (Note 2)               |
| t <sub>AFC2</sub>  | Addr Float to $\overline{PSEN}$  | 1/30 t <sub>CY</sub> -40         | 10                 |      | ns   | (Note 2)               |
| t <sub>LAFC1</sub> | ALE to Control, ( $\overline{RD}$ , $\overline{WR}$ )                    | 1/5 t <sub>CY</sub> -75          | 200                |      | ns   |                        |
| t <sub>LAFC2</sub> | ALE to Control ( $\overline{PSEN}$ )                                     | 1/10 t <sub>CY</sub> -75         | 60                 |      | ns   |                        |
| t <sub>CA1</sub>   | Control to ALE ( $\overline{RD}$ , $\overline{WR}$ , $\overline{PROG}$ ) | 1/15 t <sub>CY</sub> -40         | 50                 |      | ns   |                        |
| t <sub>CA2</sub>   | Control to ALE ( $\overline{PSEN}$ )                                     | 4/15 t <sub>CY</sub> -40         | 320                |      | ns   |                        |
| t <sub>CP</sub>    | Port Control Setup to $\overline{PROG}$                                  | 2/15 t <sub>CY</sub> -80         | 100                |      | ns   |                        |
| t <sub>PC</sub>    | Port Control Hold to $\overline{PROG}$                                   | 4/15 t <sub>CY</sub> -200        | 160                |      | ns   |                        |
| t <sub>PR</sub>    | $\overline{PROG}$ to P2 Input Valid                                      | 17/30 t <sub>CY</sub> -120       |                    | 650  | ns   |                        |
| t <sub>PF</sub>    | Input Data Hold from $\overline{PROG}$                                   | 1/10 t <sub>CY</sub>             | 0                  | 140  | ns   |                        |
| t <sub>DP</sub>    | Output Data Setup  | 2/5 t <sub>CY</sub> -150         | 400                |      | ns   |                        |
| t <sub>PD</sub>    | Output Data Hold   | 1/10 t <sub>CY</sub> -50         | 90                 |      | ns   |                        |
| t <sub>PP</sub>    | $\overline{PROG}$ Pulse Width  | 7/10 t <sub>CY</sub> -250        | 700                |      | ns   |                        |
| t <sub>PL</sub>    | Port 2 I/O Setup to ALE  | 4/15 t <sub>CY</sub> -200        | 160                |      | ns   |                        |
| t <sub>LP</sub>    | Port 2 I/O Hold to ALE   | 1/30 t <sub>CY</sub> -30         | 15                 |      | ns   |                        |
| t <sub>PV</sub>    | Port Output from ALE   | 3/10 t <sub>CY</sub> +100        |                    | 510  | ns   |                        |
| t <sub>OPRR</sub>  | T0 Rep Rate  | 3/15 t <sub>CY</sub>             | 270                |      | ns   |                        |

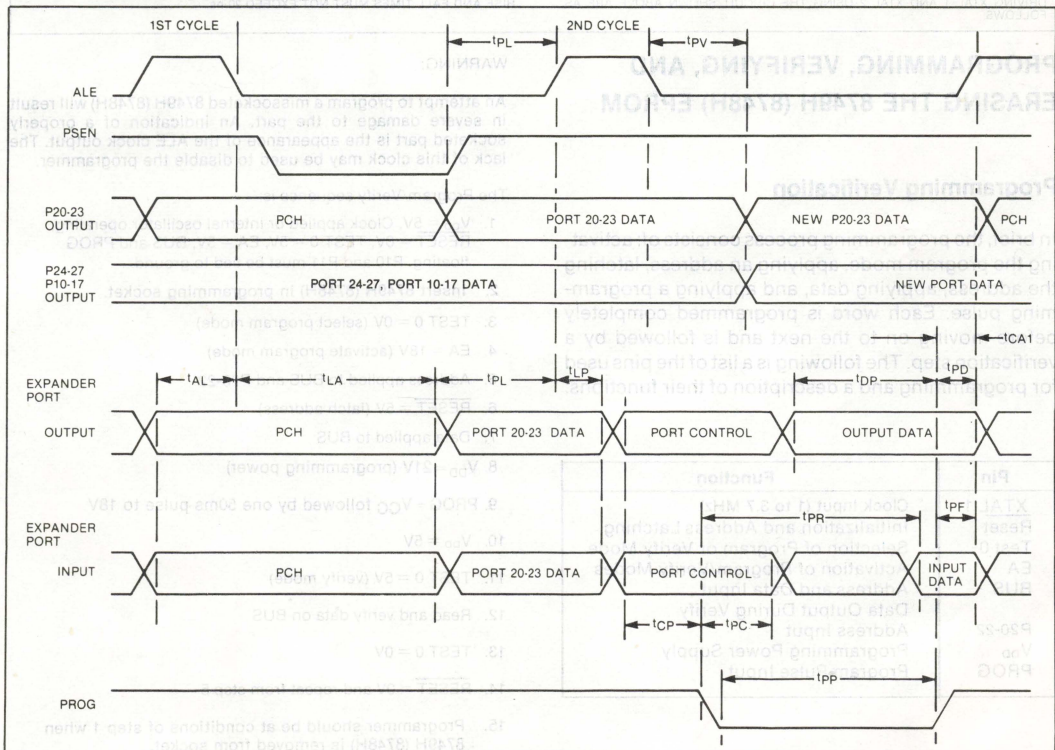
**Notes:**

- Control outputs CL = 80 pF  
BUS Outputs CL = 150 pF
- BUS High Impedance  
Load 20 pF
- 8048H/8035HL max  
Frequency = 8 MHz;  
8048H-1/8035HL-1 max  
Frequency = 11 MHz
- f(t<sub>CY</sub>) assumes 50% duty cycle  
on X1 and X2.

## WAVEFORMS

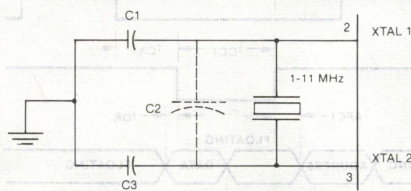


## PORT 1/PORT 2 TIMING





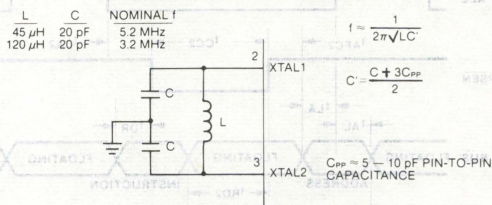
### CRYSTAL OSCILLATOR MODE



C1 = 5 pF ± ½ pF + (STRAY < 5 pF)  
C2 = (CRYSTAL + STRAY) < 8 pF  
C3 = 20 pF ± 1 pF + (STRAY < 5 pF)

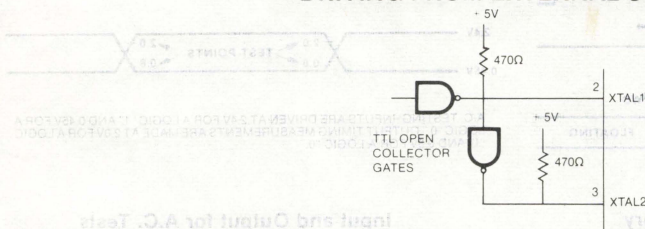
CRYSTAL SERIES RESISTANCE SHOULD BE LESS THAN 75 Ω AT 6 MHz; LESS THAN 180 Ω AT 3.6 MHz

### LC OSCILLATOR MODE



EACH C SHOULD BE APPROXIMATELY 20 pF, INCLUDING STRAY CAPACITANCE

### DRIVING FROM EXTERNAL SOURCE



FOR XTAL 1 AND XTAL 2 DEFINE "HIGH" AS VOLTAGES ABOVE 1.6V AND "LOW" AS VOLTAGES BELOW 1.6V. THE DUTY CYCLE REQUIREMENTS FOR EXTERNALLY DRIVING XTAL 1 AND XTAL 2 USING THE CIRCUIT SHOWN ABOVE ARE AS FOLLOWS:

FOR THE 8749H (8748H), XTAL 1 MUST BE HIGH 35 TO 65% OF THE PERIOD AND XTAL 2 MUST BE HIGH 35 TO 65% OF THE PERIOD. RISE AND FALL TIMES MUST NOT EXCEED 20 ns.

## PROGRAMMING, VERIFYING, AND ERASING THE 8749H (8748H) EPROM

### Programming Verification

In brief, the programming process consists of: activating the program mode, applying an address, latching the address, applying data, and applying a programming pulse. Each word is programmed completely before moving on to the next and is followed by a verification step. The following is a list of the pins used for programming and a description of their functions:

| Pin             | Function                            |
|-----------------|-------------------------------------|
| XTAL 1          | Clock Input (1 to 3.7 MHz)          |
| Reset           | Initialization and Address Latching |
| Test 0          | Selection of Program or Verify Mode |
| EA              | Activation of Program/Verify Modes  |
| BUS             | Address and Data Input              |
|                 | Data Output During Verify           |
| V <sub>DD</sub> | Address Input                       |
|                 | Programming Power Supply            |
|                 | Program Pulse Input                 |

### WARNING:

An attempt to program a missocketed 8749H (8748H) will result in severe damage to the part. An indication of a properly socketed part is the appearance of the ALE clock output. The lack of this clock may be used to disable the programmer.

The Program/Verify sequence is:

1. V<sub>DD</sub> = 5V, Clock applied or internal oscillator operating, RESET = 0V, TEST 0 = 5V, EA = 5V, BUS and PROG floating. P10 and P11 must be tied to ground.
2. Insert 8749H (8748H) in programming socket.
3. TEST 0 = 0V (select program mode)
4. EA = 18V (activate program mode)
5. Address applied to BUS and P20-22
6. RESET = 5V (latch address)
7. Data applied to BUS
8. V<sub>DD</sub> = 21V (programming power)
9. PROG = V<sub>CC</sub> followed by one 50ms pulse to 18V
10. V<sub>DD</sub> = 5V
11. TEST 0 = 5V (verify mode)
12. Read and verify data on BUS
13. TEST 0 = 0V
14. RESET = 0V and repeat from step 5
15. Programmer should be at conditions of step 1 when 8749H (8748H) is removed from socket.

**A.C. TIMING SPECIFICATION FOR PROGRAMMING** ( $T_A = 25^\circ\text{C} \pm 5^\circ\text{C}$ ,  $V_{CC} = 5\text{V} \pm 5\%$ ,  $V_{DD} = 21 \pm .5\text{V}$ )  
**8748H/8749H ONLY**

| Symbol     | Parameter  | Min       | Max       | Unit          | Test Conditions |
|------------|--|-----------|-----------|---------------|-----------------|
| $t_{AW}$   | Address Setup Time to $\overline{\text{RESET}}$ †      | $4t_{CY}$ |           |               |                 |
| $t_{WA}$   | Address Hold Time After $\overline{\text{RESET}}$ †    | $4t_{CY}$ |           |               |                 |
| $t_{DW}$   | Data in Setup Time to PROG †                           | $4t_{CY}$ |           |               |                 |
| $t_{WD}$   | Data in Hold Time After PROG †                         | $4t_{CY}$ |           |               |                 |
| $t_{PH}$   | $\overline{\text{RESET}}$ Hold Time to Verify          | $4t_{CY}$ |           |               |                 |
| $t_{VDDW}$ | $V_{DD}$ Hold Time Before PROG †                       | 0         | 1.0       | mS            |                 |
| $t_{VDDH}$ | $V_{DD}$ Hold Time After PROG †                        | 0         | 1.0       | mS            |                 |
| $t_{PW}$   | Program Pulse Width                                    | 50        | 60        | mS            |                 |
| $t_{TW}$   | Test 0 Setup Time for Program Mode                     | $4t_{CY}$ |           |               |                 |
| $t_{WT}$   | Test 0 Hold Time After Program Mode                    | $4t_{CY}$ |           |               |                 |
| $t_{DO}$   | Test 0 to Data Out Delay                               |           | $4t_{CY}$ |               |                 |
| $t_{WW}$   | $\overline{\text{RESET}}$ Pulse Width to Latch Address | $4t_{CY}$ |           |               |                 |
| $t_r, t_f$ | $V_{DD}$ and PROG Rise and Fall Times                  | 0.5       | 100       | $\mu\text{s}$ |                 |
| $t_{CY}$   | CPU Operation Cycle Time                               | 4.0       | 15        | $\mu\text{s}$ |                 |
| $t_{RE}$   | $\overline{\text{RESET}}$ Setup Time before EA†        | $4t_{CY}$ |           |               |                 |

NOTE: If Test 0 is high  $t_{DO}$  can be triggered by  $\overline{\text{RESET}}$  †

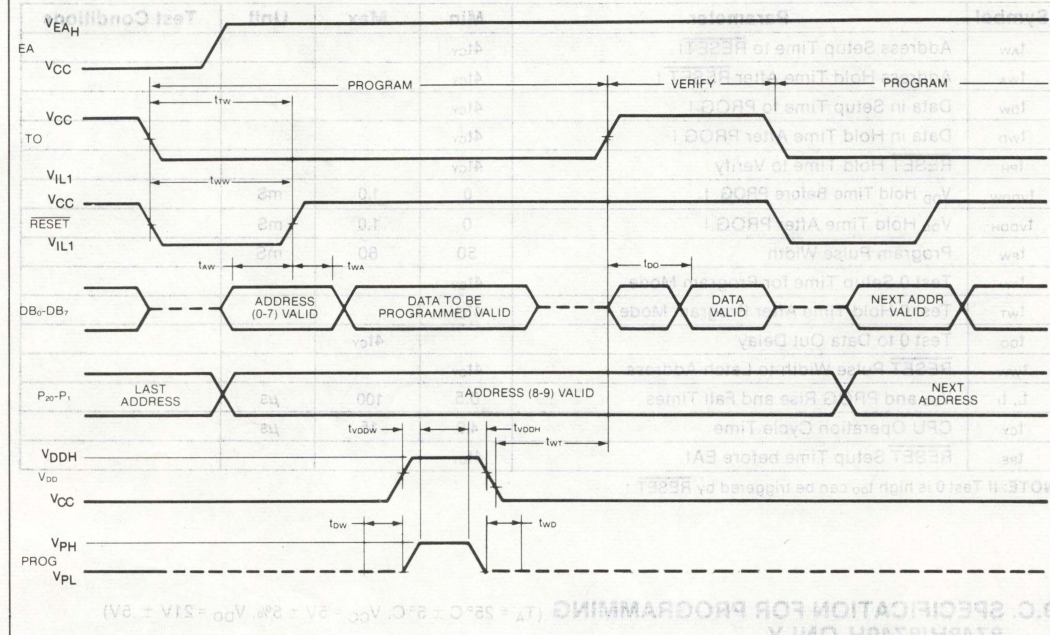
**D.C. SPECIFICATION FOR PROGRAMMING** ( $T_A = 25^\circ\text{C} \pm 5^\circ\text{C}$ ,  $V_{CC} = 5\text{V} \pm 5\%$ ,  $V_{DD} = 21\text{V} \pm .5\text{V}$ )  
**8748H/8749H ONLY**

| Symbol     | Parameter                               | Min  | Max      | Unit | Test Conditions |
|------------|---|------|----------|------|-----------------|
| $V_{DDH}$  | $V_{DD}$ Program Voltage High Level     | 20.5 | 21.5     | V    |                 |
| $V_{DDL}$  | $V_{DD}$ Voltage Low Level              | 4.75 | 5.25     | V    |                 |
| $V_{PH}$   | PROG Program Voltage High Level         | 17.5 | 18.5     | V    |                 |
| $V_{PL}$   | PROG Voltage Low Level                  | 4.0  | $V_{CC}$ | V    |                 |
| $V_{EAH}$  | EA Program or Verify Voltage High Level | 17.5 | 18.5     | V    |                 |
| $I_{DD}$   | $V_{DD}$ High Voltage Supply Current    |      | 20.0     | mA   |                 |
| $I_{PROG}$ | PROG High Voltage Supply Current        |      | 1.0      | mA   |                 |
| $I_{EA}$   | EA High Voltage Supply Current          |      | 1.0      | mA   |                 |



## WAVEFORMS

### COMBINATION PROGRAM/VERIFY MODE (EPROM's ONLY)



| Symbol          | Parameter                  | Unit | Max  | Min  | Test Conditions |
|-----------------|----------------------------|------|------|------|-----------------|
| V <sub>DD</sub> | Program Voltage High Level | V    | 21.5 | 20.5 |                 |
| V <sub>DD</sub> | Program Voltage Low Level  | V    | 5.25 | 4.75 |                 |
| V <sub>PP</sub> | Program Voltage High Level | V    | 18.5 | 17.5 |                 |
| V <sub>PP</sub> | Program Voltage Low Level  | V    | 5.0  | 4.0  |                 |
| I <sub>DD</sub> | Program Supply Current     | mA   | 20.0 | 17.5 |                 |

**VERIFY MODE (ROM/EPROM)**

TO, RESET

DB<sub>7</sub>-DB<sub>0</sub>

ADDRESS (0-7) VALID

DATA OUT VALID

NEXT ADDRESS

NEXT DATA OUT VALID

P<sub>20</sub>-P<sub>1</sub>

ADDRESS (8-9) VALID

NEXT ADDRESS VALID

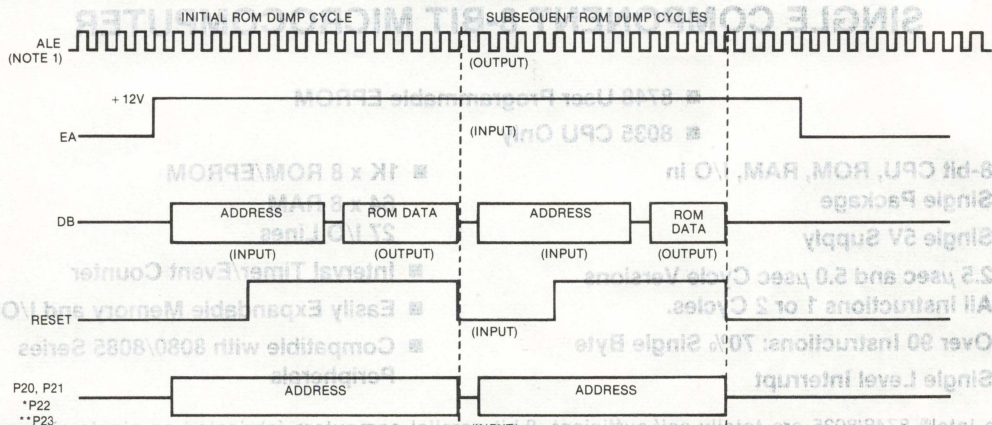
**NOTES:**

1. PROG MUST FLOAT IF EA IS LOW (i.e.,  $\neq 23V$ ).
2. X<sub>1</sub> AND X<sub>2</sub> DRIVEN BY 3 MHz CLOCK WILL GIVE 5 $\mu$ sec t<sub>cy</sub>. THIS IS ACCEPTABLE FOR — 8 PARTS AS WELL AS STANDARD PARTS.

The 8748 EPROM can be programmed by either of two Intel products:

- OMPT-48 Microcomputer Design Aid, or  
 versal PROM Programmer (UPP series) per-  
 the Intellec® Development System with a  
 sonality Card.

# SUGGESTED ROM VERIFICATION ALGORITHM FOR H-MOS DEVICES ONLY



P22 = 0V (8048H)

P23 = 0V (8048H/8049H)

8049H/8050AH

\*\*8050AH

VCC = VDD = +5V

VSS = 0V

NOTE 1: ALE IS FUNCTION OF X1, X2 INPUTS. PLEASE REFER TO FIGURE ON PAGE 10-20 FOR OSCILLATOR CONFIGURATIONS

These microcomputers are designed to be efficient controllers as well as arithmetic processors. They have extensive bit handling capability as well as facilities for both binary and BCD arithmetic. Efficient use of program memory results from an instruction set consisting mostly of single bit instructions and no instructions over 2 bytes in length.

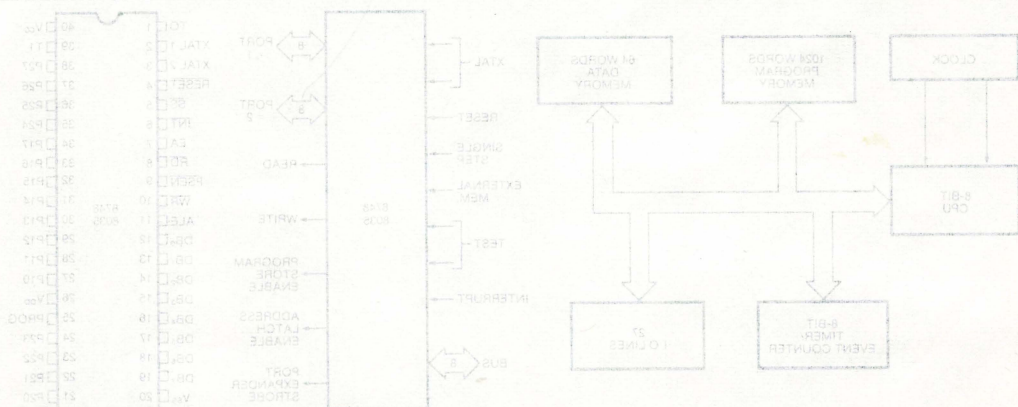


Figure 3. Pin Configuration

Figure 2. Logic Symbol

Figure 1. Block Diagram



# 8748/8035 SINGLE COMPONENT 8-BIT MICROCOMPUTER

- 8748 User Programmable EPROM
- 8035 CPU Only
- 8-bit CPU, ROM, RAM, I/O in Single Package
- Single 5V Supply
- 2.5  $\mu$ sec and 5.0  $\mu$ sec Cycle Versions All Instructions 1 or 2 Cycles.
- Over 90 Instructions: 70% Single Byte
- Single Level Interrupt
- 1K x 8 ROM/EPROM
- 64 x 8 RAM
- 27 I/O Lines
- Interval Timer/Event Counter
- Easily Expandable Memory and I/O
- Compatible with 8080/8085 Series Peripherals

The Intel® 8748/8035 are totally self-sufficient, 8-bit parallel computers fabricated on single silicon chips using Intel's N-channel silicon gate MOS process.

The 8748 contains a 1K x 8 UV-erasable, user-programmable program memory, a 64x8 RAM data memory, 27 I/O lines, and an 8-bit timer/counter in addition to on-board oscillator and clock circuits. For systems that require extra capability, the 8748 can be expanded using standard memories and MCS-80®/MCS-85® peripherals. The 8035 is the equivalent of an 8748 without program memory and can be used with external ROM and RAM. To reduce development problems to a minimum and provide maximum flexibility, three interchangeable pin-compatible versions of this single component microcomputer exist: the 8748 with user-programmable and erasable EPROM program memory, the 8048 with factory-programmed mask ROM program memory for low cost, high volume production, and the 8035 without program memory for use with external program memories.

These microcomputers are designed to be efficient controllers as well as arithmetic processors. They have extensive bit handling capability as well as facilities for both binary and BCD arithmetic. Efficient use of program memory results from an instruction set consisting mostly of single bit instructions and no instructions over 2 bytes in length.

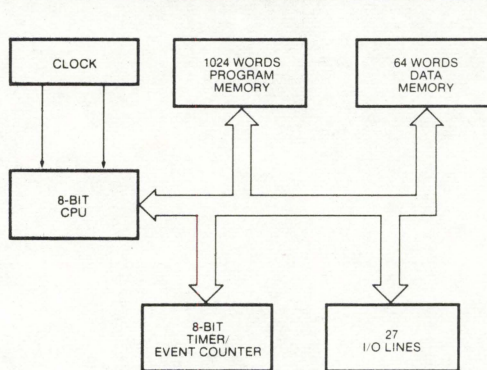


Figure 1. Block Diagram

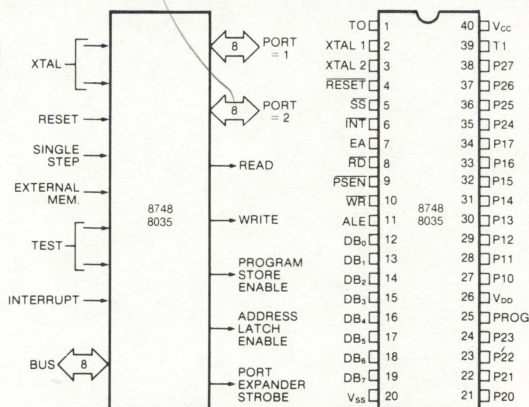


Figure 2. Logic Symbol

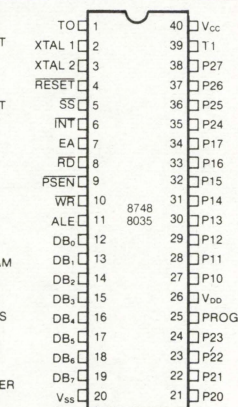


Figure 3. Pin Configuration



Table 1. Pin Description

| Symbol                               | Pin No. | Function  |
|--------------------------------------|---------|---|
| V <sub>SS</sub>                      | 20      | Circuit GND Potential   |
| V <sub>DD</sub>                      | 26      | Programming power supply; +25V during program, +5V during operation.  |
| V <sub>CC</sub>                      | 40      | Main power supply; +5V during operation and programming.  |
| PROG                                 | 25      | Program pulse (+23V) input pin during 8748 programming.<br>Output strobe for 8243 I/O expander.   |
| P10-P17 Port 1                       | 27-34   | 8-bit quasi-bidirectional port.   |
| P20-P27 Port 2                       | 21-24   | 8-bit quasi-bidirectional port.   |
|                                      | 35-38   | P20-P23 contain the four high order program counter bits during an external program memory fetch and serve as a 4-bit I/O expander bus for 8243.  |
| DB <sub>0</sub> -DB <sub>7</sub> BUS | 12-19   | True bidirectional port which can be written or read synchronously using the RD, WR strobes. The port can also be statically latched.<br>Contains the 8 low order program counter bits during an external program memory fetch, and receives the addressed instruction under the control of PSEN. Also contains the address and data during an external RAM data store instruction, under control of ALE, RD, and WR. |
| T0                                   | 1       | Input pin testable using the conditional transfer instructions JT0 and JNT0. T0 can be designated as a clock output using ENT0 CLK instruction. T0 is also used during programming.   |
| T1                                   | 39      | Input pin testable using the JT1, and JNT1 instructions. Can be designated the timer/counter input using the STRT CNT instruction.  |
| INT                                  | 6       | Interrupt input. Initiates an interrupt if interrupt is enabled. Interrupt is disabled after a reset. Also testable with conditional jump instruction. (Active low)   |

| Symbol | Pin No. | Function  |
|--------|---------|---|
| RD     | 8       | Output strobe activated during a BUS read. Can be used to enable data onto the bus from an external device.<br>Used as a read strobe to external data memory. (Active low)                    |
| RESET  | 4       | Input which is used to initialize the processor. Also used during PROM programming verification, and power down. (Active low) (Non TTL V <sub>IH</sub> )                                      |
| WR     | 10      | Output strobe during a bus write. (Active low)<br>Used as write strobe to external data memory.   |
| ALE    | 11      | Address latch enable. This signal occurs once during each cycle and is useful as a clock output.<br>The negative edge of ALE strobes address into external data and program memory.           |
| PSEN   | 9       | Program store enable. This output occurs only during a fetch to external program memory. (Active low)   |
| SS     | 5       | Single step input can be used in conjunction with ALE to "single step" the processor through each instruction. (Active low)   |
| EA     | 7       | External access input which forces all program memory fetches to reference external memory. Useful for emulation and debug, and essential for testing and program verification. (Active high) |
| XTAL1  | 2       | One side of crystal input for internal oscillator. Also input for external source. (Non TTL V <sub>IH</sub> )   |
| XTAL2  | 3       | Other side of crystal input.  |



Table 2. Instruction Set Summary

|               | Mnemonic       | Description                       | Bytes | Cycle |
|---------------|----------------|-----------------------------------|-------|-------|
| Accumulator   | ADD A, R       | Add register to A                 | 1     | 1     |
|               | ADD A, @R      | Add data memory to A              | 1     | 1     |
|               | ADD A, #data   | Add immediate to A                | 2     | 2     |
|               | ADDC A, R      | Add register with carry           | 1     | 1     |
|               | ADDC A, @R     | Add data memory with carry        | 1     | 1     |
|               | ADDC A, #data  | Add immediate with carry          | 2     | 2     |
|               | ANL A, R       | And register to A                 | 1     | 1     |
|               | ANL A, @R      | And data memory to A              | 1     | 1     |
|               | ANL A, #data   | And immediate to A                | 2     | 2     |
|               | ORL A, R       | Or register to A                  | 1     | 1     |
|               | ORL A, @R      | Or data memory to A               | 1     | 1     |
|               | ORL A, #data   | Or immediate to A                 | 2     | 2     |
|               | XRL A, R       | Exclusive or register to A        | 1     | 1     |
|               | XRL A, @R      | Exclusive or data memory to A     | 1     | 1     |
|               | XRL A, #data   | Exclusive or immediate to A       | 2     | 2     |
|               | INC A          | Increment A                       | 1     | 1     |
|               | DEC A          | Decrement A                       | 1     | 1     |
|               | CLR A          | Clear A                           | 1     | 1     |
|               | CPL A          | Complement A                      | 1     | 1     |
|               | DA A           | Decimal adjust A                  | 1     | 1     |
| Input/Output  | SWAP A         | Swap nibbles of A                 | 1     | 1     |
|               | RL A           | Rotate A left                     | 1     | 1     |
|               | RLC A          | Rotate A left through carry       | 1     | 1     |
|               | RR A           | Rotate A right                    | 1     | 1     |
|               | RRC A          | Rotate A right through carry      | 1     | 1     |
|               | IN A, P        | Input port to A                   | 1     | 2     |
|               | OUTL P, A      | Output A to port                  | 1     | 2     |
|               | ANL P, #data   | And immediate to port             | 2     | 2     |
|               | ORL P, #data   | Or immediate to port              | 2     | 2     |
|               | INS A, BUS     | Input BUS to A                    | 1     | 2     |
| Registers     | OUTL BUS, A    | Output A to BUS                   | 1     | 2     |
|               | ANL BUS, #data | And immediate to BUS              | 2     | 2     |
|               | ORL BUS, #data | Or immediate to BUS               | 2     | 2     |
|               | MOVD A, P      | Input expander port to A          | 1     | 2     |
|               | MOVD P, A      | Output A to expander port         | 1     | 2     |
|               | ANLD P, A      | And A to expander port            | 1     | 2     |
|               | ORLD P, A      | Or A to expander port             | 1     | 2     |
|               | INC R          | Increment register                | 1     | 1     |
|               | INC @R         | Increment data memory             | 1     | 1     |
|               | DEC R          | Decrement register                | 1     | 1     |
| Branch        | JMP addr       | Jump unconditional                | 2     | 2     |
|               | JMPP @A        | Jump indirect                     | 1     | 2     |
|               | DJNZ R, addr   | Decrement register and skip       | 2     | 2     |
|               | JC addr        | Jump on carry = 1                 | 2     | 2     |
|               | JNC addr       | Jump on carry = 0                 | 2     | 2     |
|               | JZ addr        | Jump on A zero                    | 2     | 2     |
|               | JNZ addr       | Jump on A not zero                | 2     | 2     |
|               | JT0 addr       | Jump on T0 = 1                    | 2     | 2     |
|               | JNT0 addr      | Jump on T0 = 0                    | 2     | 2     |
|               | JT1 addr       | Jump on T1 = 1                    | 2     | 2     |
|               | JNT1 addr      | Jump on T1 = 0                    | 2     | 2     |
|               | JF0 addr       | Jump on F0 = 1                    | 2     | 2     |
|               | JF1 addr       | Jump on F1 = 1                    | 2     | 2     |
|               | JTF addr       | Jump on timer flag                | 2     | 2     |
|               | JNI addr       | Jump on INT = 0                   | 2     | 2     |
|               | JBb addr       | Jump on accumulator bit           | 2     | 2     |
| Subroutine    | CALL addr      | Jump to subroutine                | 2     | 2     |
|               | RET            | Return                            | 1     | 2     |
|               | RETR           | Return and restore status         | 1     | 2     |
| Flags         | CLR C          | Clear carry                       | 1     | 1     |
|               | CPL C          | Complement carry                  | 1     | 1     |
|               | CLR F0         | Clear flag 0                      | 1     | 1     |
|               | CPL F0         | Complement flag 0                 | 1     | 1     |
|               | CLR F1         | Clear flag 1                      | 1     | 1     |
| Data Moves    | CPL F1         | Complement flag 1                 | 1     | 1     |
|               | MOV A, R       | Move register to A                | 1     | 1     |
|               | MOV A, @R      | Move data memory to A             | 1     | 1     |
|               | MOV A, #data   | Move immediate to A               | 2     | 2     |
|               | MOV R, A       | Move A to register                | 1     | 1     |
|               | MOV @R, A      | Move A to data memory             | 1     | 1     |
|               | MOV R, #data   | Move immediate to register        | 2     | 2     |
|               | MOV @R, #data  | Move immediate to data memory     | 2     | 2     |
|               | MOV A, PSW     | Move PSW to A                     | 1     | 1     |
|               | MOV PSW, A     | Move A to PSW                     | 1     | 1     |
|               | XCH A, R       | Exchange A and register           | 1     | 1     |
|               | XCH A, @R      | Exchange A and data memory        | 1     | 1     |
|               | XCHD A, @R     | Exchange nibble of A and register | 1     | 1     |
|               | MOVX A, @R     | Move external data memory to A    | 1     | 2     |
|               | MOVX @R, A     | Move A to external data memory    | 1     | 2     |
|               | MOVP A, @A     | Move to A from current page       | 1     | 2     |
|               | MOVP3 A, @A    | Move to A from page 3             | 1     | 2     |
| Timer/Counter | MOV A, T       | Read timer/counter                | 1     | 1     |
|               | MOV T, A       | Load timer/counter                | 1     | 1     |
|               | STRT T         | Start timer                       | 1     | 1     |
|               | STRT CNT       | Start counter                     | 1     | 1     |
|               | STOP TCNT      | Stop timer/counter                | 1     | 1     |
|               | EN TCNTI       | Enable timer/counter interrupt    | 1     | 1     |
|               | DIS TCNTI      | Disable timer/counter interrupt   | 1     | 1     |
| Control       | EN I           | Enable external interrupt         | 1     | 1     |
|               | DIS I          | Disable external interrupt        | 1     | 1     |
|               | SEL RB0        | Select register bank 0            | 1     | 1     |
|               | SEL RB1        | Select register bank 1            | 1     | 1     |
|               | SEL MB0        | Select memory bank 0              | 1     | 1     |
|               | SEL MB1        | Select memory bank 1              | 1     | 1     |
|               | ENT0 CLK       | Enable clock output on T0         | 1     | 1     |
| NOP           | NOP            | No operation                      | 1     | 1     |

Mnemonics copyright Intel Corporation 1982

# ABSOLUTE MAXIMUM RATINGS\*

Ambient Temperature Under Bias ..... 0°C to 70°C  
 Storage Temperature ..... - 65°C to +125°C  
 Voltage on Any Pin with Respect  
 to Ground ..... - 0.5V to +7V  
 Power Dissipation ..... 1.5 Watt

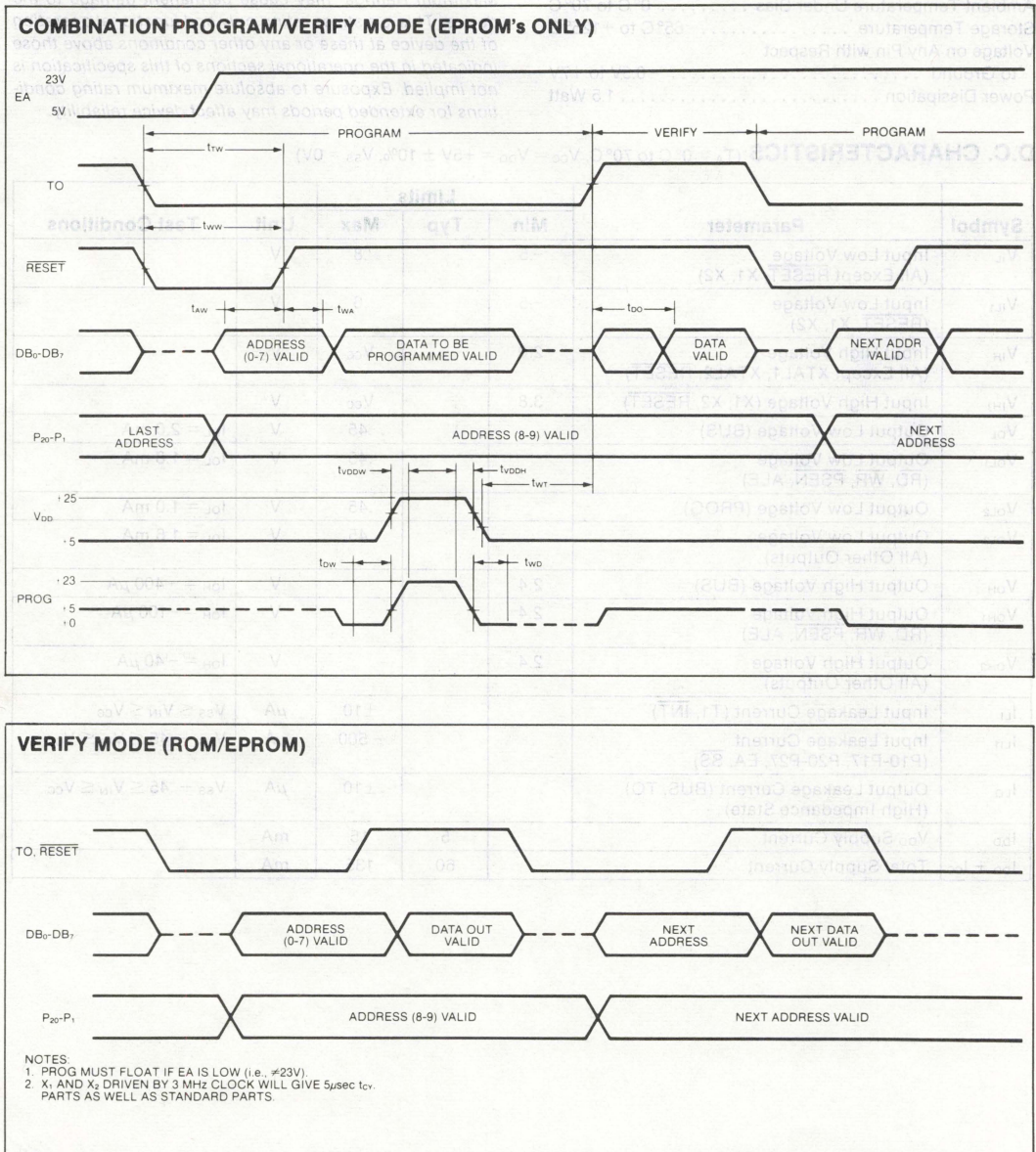
\*NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

# D.C. CHARACTERISTICS (T<sub>A</sub> = 0°C to 70°C, V<sub>CC</sub> = V<sub>DD</sub> = +5V ± 10%, V<sub>SS</sub> = 0V)

| Symbol                            | Parameter  | Limits |     |                 | Unit | Test Conditions   |
|-----------------------------------|--|--------|-----|-----------------|------|---|
|                                   |  | Min    | Typ | Max             |      |   |
| V <sub>IL</sub>                   | Input Low Voltage<br>(All Except RESET, X1, X2)            | -.5    |     | .8              | V    |   |
| V <sub>IL1</sub>                  | Input Low Voltage<br>(RESET, X1, X2)                       | -.5    |     | .6              | V    |   |
| V <sub>IH</sub>                   | Input High Voltage<br>(All Except XTAL1, XTAL2, RESET)     | 2.0    |     | V <sub>CC</sub> | V    |   |
| V <sub>IH1</sub>                  | Input High Voltage (X1, X2, RESET)                         | 3.8    |     | V <sub>CC</sub> | V    |   |
| V <sub>OL</sub>                   | Output Low Voltage (BUS)                                   |        |     | .45             | V    | I <sub>OL</sub> = 2.0 mA                                  |
| V <sub>OL1</sub>                  | Output Low Voltage<br>(RD, WR, PSEN, ALE)                  |        |     | .45             | V    | I <sub>OL</sub> = 1.8 mA                                  |
| V <sub>OL2</sub>                  | Output Low Voltage (PROG)                                  |        |     | .45             | V    | I <sub>OL</sub> = 1.0 mA                                  |
| V <sub>OL3</sub>                  | Output Low Voltage<br>(All Other Outputs)                  |        |     | .45             | V    | I <sub>OL</sub> = 1.6 mA                                  |
| V <sub>OH</sub>                   | Output High Voltage (BUS)                                  | 2.4    |     |                 | V    | I <sub>OH</sub> = - 400 µA                                |
| V <sub>OH1</sub>                  | Output High Voltage<br>(RD, WR, PSEN, ALE)                 | 2.4    |     |                 | V    | I <sub>OH</sub> = -100 µA                                 |
| V <sub>OH2</sub>                  | Output High Voltage<br>(All Other Outputs)                 | 2.4    |     |                 | V    | I <sub>OH</sub> = - 40 µA                                 |
| I <sub>LI</sub>                   | Input Leakage Current (T1, INT)                            |        |     | ±10             | µA   | V <sub>SS</sub> ≤ V <sub>IN</sub> ≤ V <sub>CC</sub>       |
| I <sub>LI1</sub>                  | Input Leakage Current<br>(P10-P17, P20-P27, EA, SS)        |        |     | - 500           | µA   | V <sub>SS</sub> + .45 ≤ V <sub>IN</sub> ≤ V <sub>CC</sub> |
| I <sub>LO</sub>                   | Output Leakage Current (BUS, TO)<br>(High Impedance State) |        |     | ±10             | µA   | V <sub>SS</sub> + .45 ≤ V <sub>IN</sub> ≤ V <sub>CC</sub> |
| I <sub>DD</sub>                   | V <sub>DD</sub> Supply Current                             |        | 5   | 15              | mA   |   |
| I <sub>DD</sub> + I <sub>CC</sub> | Total Supply Current                                       |        | 60  | 135             | mA   |   |



# WAVEFORMS



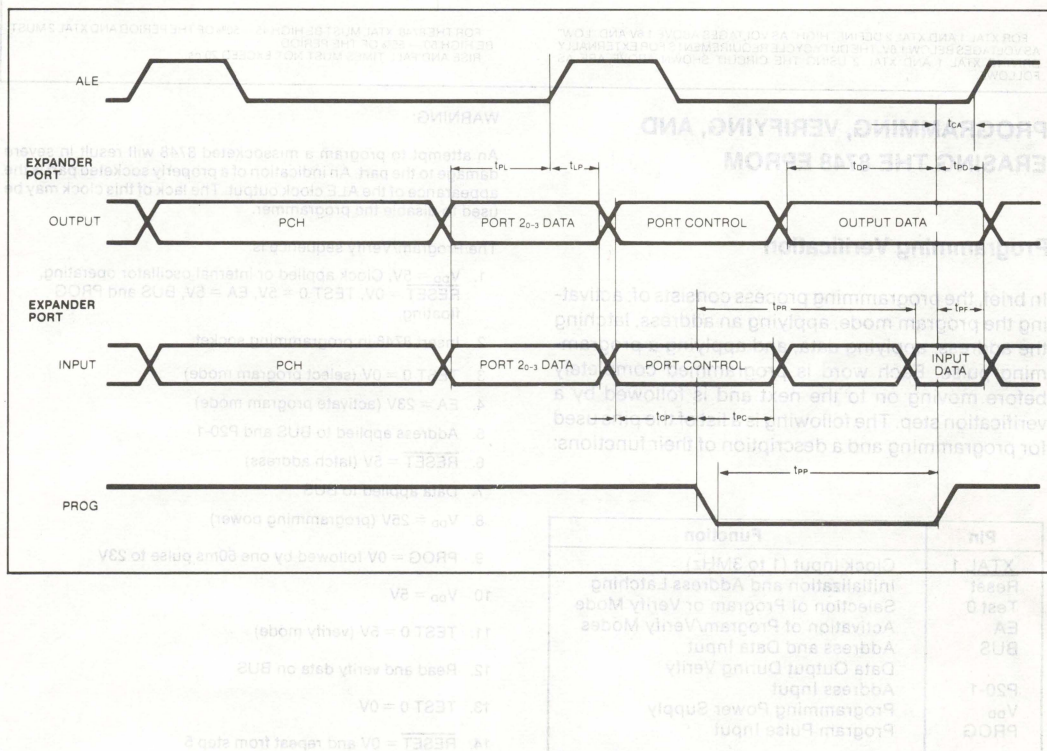
The 8748 EPROM can be programmed by either of two Intel products:

1. PROMPT-48 Microcomputer Design Aid, or
2. Universal PROM Programmer (UPP series) peripheral of the Intellec® Development System with a UPP-848 Personality Card.

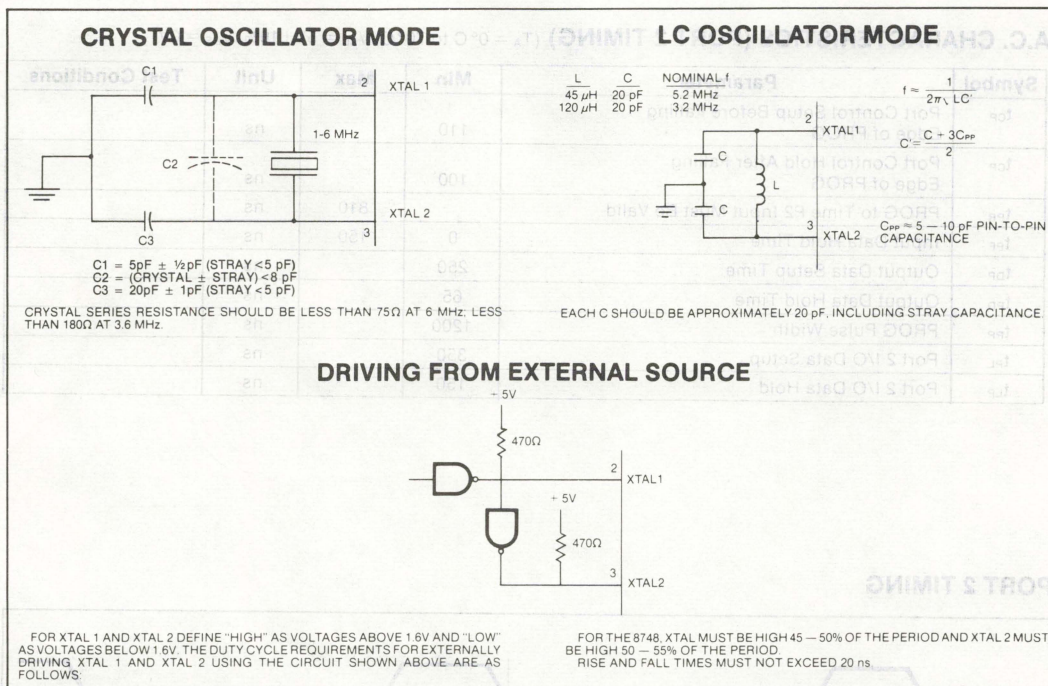
# A.C. CHARACTERISTICS (PORT 2 TIMING) ( $T_A = 0^\circ\text{C}$ to $70^\circ\text{C}$ , $V_{CC} = 5V \pm 10\%$ , $V_{SS} = 0V$ )

| Symbol   | Parameter                                      | Min  | Max | Unit | Test Conditions |
|----------|--|------|-----|------|-----------------|
| $t_{CP}$ | Port Control Setup Before Falling Edge of PROG | 110  |     | ns   |                 |
| $t_{CH}$ | Port Control Hold After Falling Edge of PROG   | 100  |     | ns   |                 |
| $t_{PR}$ | PROG to Time P2 Input Must Be Valid            |      | 810 | ns   |                 |
| $t_{PF}$ | Input Data Hold Time                           | 0    | 150 | ns   |                 |
| $t_{DP}$ | Output Data Setup Time                         | 250  |     | ns   |                 |
| $t_{DH}$ | Output Data Hold Time                          | 65   |     | ns   |                 |
| $t_{PP}$ | PROG Pulse Width                               | 1200 |     | ns   |                 |
| $t_{PL}$ | Port 2 I/O Data Setup                          | 350  |     | ns   |                 |
| $t_{LP}$ | Port 2 I/O Data Hold                           | 150  |     | ns   |                 |

## PORT 2 TIMING







## PROGRAMMING, VERIFYING, AND ERASING THE 8748 EPROM

### Programming Verification

In brief, the programming process consists of: activating the program mode, applying an address, latching the address, applying data, and applying a programming pulse. Each word is programmed completely before moving on to the next and is followed by a verification step. The following is a list of the pins used for programming and a description of their functions:

| Pin             | Function                            |
|-----------------|-------------------------------------|
| XTAL 1          | Clock Input (1 to 3MHz)             |
| Reset           | Initialization and Address Latching |
| Test 0          | Selection of Program or Verify Mode |
| EA              | Activation of Program/Verify Modes  |
| BUS             | Address and Data Input              |
| P20-1           | Data Output During Verify           |
| V <sub>DD</sub> | Address Input                       |
| PROG            | Programming Power Supply            |
|                 | Program Pulse Input                 |

### WARNING:

An attempt to program a missocketed 8748 will result in severe damage to the part. An indication of a properly socketed part is the appearance of the ALE clock output. The lack of this clock may be used to disable the programmer.

The Program/Verify sequence is:

1. V<sub>DD</sub> = 5V, Clock applied or internal oscillator operating, RESET = 0V, TEST 0 = 5V, EA = 5V, BUS and PROG floating.
2. Insert 8748 in programming socket.
3. TEST 0 = 0V (select program mode)
4. EA = 23V (activate program mode)
5. Address applied to BUS and P20-1
6. RESET = 5V (latch address)
7. Data applied to BUS
8. V<sub>DD</sub> = 25V (programming power)
9. PROG = 0V followed by one 50ms pulse to 23V
10. V<sub>DD</sub> = 5V
11. TEST 0 = 5V (verify mode)
12. Read and verify data on BUS
13. TEST 0 = 0V
14. RESET = 0V and repeat from step 5
15. Programmer should be at conditions of step 1 when 8748 is removed from socket.

# A.C. TIMING SPECIFICATION FOR PROGRAMMING

( $T_A = 25^\circ\text{C} \pm 5^\circ\text{C}$ ,  $V_{CC} = 5\text{V} \pm 5\%$ ,  $V_{DD} = 25\text{V} \pm 1\text{V}$ )

| Symbol     | Parameter  | Min       | Max       | Unit          | Test Conditions |
|------------|--|-----------|-----------|---------------|-----------------|
| $t_{AW}$   | Address Setup Time to $\overline{\text{RESET}}$        | $4t_{CY}$ |           |               |                 |
| $t_{WA}$   | Address Hold Time After $\overline{\text{RESET}}$      | $4t_{CY}$ |           |               |                 |
| $t_{DW}$   | Data in Setup Time to PROG                             | $4t_{CY}$ |           |               |                 |
| $t_{WD}$   | Data in Hold Time After PROG                           | $4t_{CY}$ |           |               |                 |
| $t_{PH}$   | $\overline{\text{RESET}}$ Hold Time to Verify          | $4t_{CY}$ |           |               |                 |
| $t_{VDDW}$ | $V_{DD}$   | $4t_{CY}$ |           |               |                 |
| $t_{VDDH}$ | $V_{DD}$ Hold Time After PROG                          | 0         |           |               |                 |
| $t_{PW}$   | Program Pulse Width                                    | 50        | 60        | mS            |                 |
| $t_{TW}$   | Test 0 Setup Time for Program Mode                     | $4t_{CY}$ |           |               |                 |
| $t_{WT}$   | Test 0 Hold Time After Program Mode                    | $4t_{CY}$ |           |               |                 |
| $t_{DO}$   | Test 0 to Data Out Delay                               |           | $4t_{CY}$ |               |                 |
| $t_{WW}$   | $\overline{\text{RESET}}$ Pulse Width to Latch Address | $4t_{CY}$ |           |               |                 |
| $t_r, t_f$ | $V_{DD}$ and PROG Rise and Fall Times                  | 0.5       | 2.0       | $\mu\text{S}$ |                 |
| $t_{CY}$   | CPU Operation Cycle Time                               | 5.0       |           | $\mu\text{S}$ |                 |
| $t_{RE}$   | $\overline{\text{RESET}}$ Setup Time before EA         | $4t_{CY}$ |           |               |                 |

NOTE: If Test 0 is high  $t_{DO}$  can be triggered by  $\overline{\text{RESET}}$

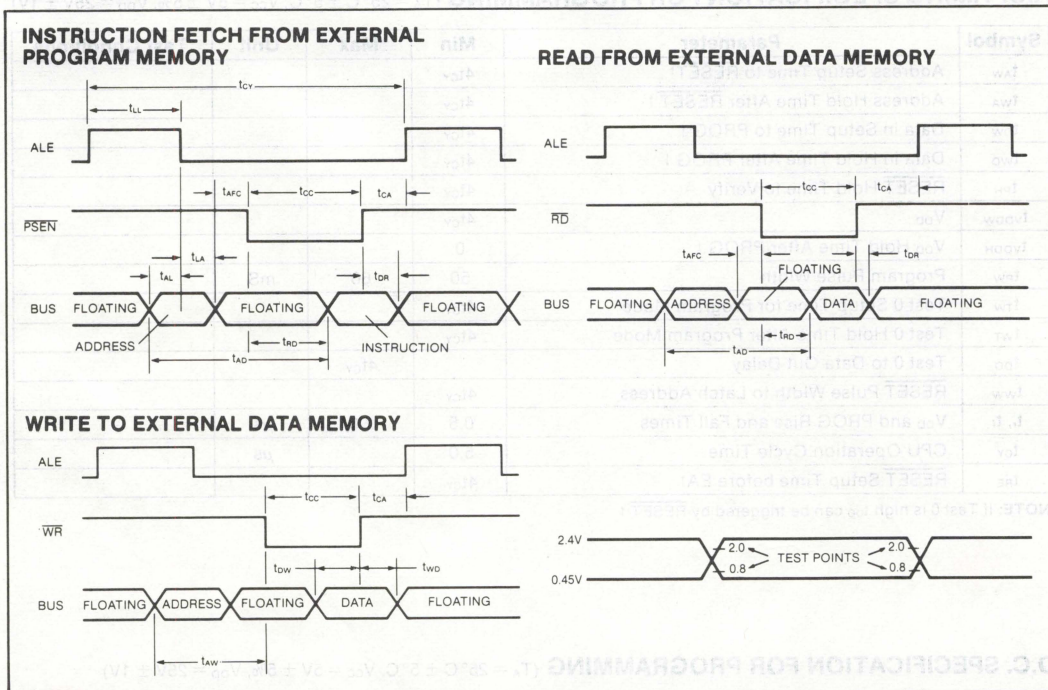
# D.C. SPECIFICATION FOR PROGRAMMING

( $T_A = 25^\circ\text{C} \pm 5^\circ\text{C}$ ,  $V_{CC} = 5\text{V} \pm 5\%$ ,  $V_{DD} = 25\text{V} \pm 1\text{V}$ )

| Symbol     | Parameter                               | Min  | Max  | Unit | Test Conditions |
|------------|---|------|------|------|-----------------|
| $V_{DOH}$  | $V_{DD}$ Program Voltage High Level     | 24.0 | 26.0 | V    |                 |
| $V_{DDL}$  | $V_{DD}$ Voltage Low Level              | 4.75 | 5.25 | V    |                 |
| $V_{PH}$   | PROG Program Voltage High Level         | 21.5 | 24.5 | V    |                 |
| $V_{PL}$   | PROG Voltage Low Level                  |      | 0.2  | V    |                 |
| $V_{EAH}$  | EA Program or Verify Voltage High Level | 21.5 | 24.5 | V    | 8748            |
| $V_{EAL}$  | EA Voltage Low Level                    |      | 5.25 | V    |                 |
| $I_{DD}$   | $V_{DD}$ High Voltage Supply Current    |      | 30.0 | mA   |                 |
| $I_{PROG}$ | PROG High Voltage Supply Current        |      | 16.0 | mA   |                 |
| $I_{EA}$   | EA High Voltage Supply Current          |      | 1.0  | mA   |                 |



## WAVEFORMS



## A.C. CHARACTERISTICS (T<sub>A</sub> = 0°C to 70°C\*, V<sub>CC</sub> = V<sub>DD</sub> = +5V ± 10%, V<sub>SS</sub> = 0V)

| Symbol           | Parameter                          | 8748/8035 |      | Unit | Conditions (Note 1)   |
|------------------|------------------------------------|-----------|------|------|-----------------------|
|                  |                                    | Min       | Max  |      |                       |
| t <sub>LL</sub>  | ALE Pulse Width                    | 400       |      | ns   |                       |
| t <sub>AL</sub>  | Address Setup to ALE               | 120       |      | ns   |                       |
| t <sub>LA</sub>  | Address Hold from ALE              | 80        |      | ns   |                       |
| t <sub>CC</sub>  | Control Pulse Width (PSEN, RD, WR) | 700       |      | ns   |                       |
| t <sub>DW</sub>  | Data Setup before WR               | 500       |      | ns   |                       |
| t <sub>WD</sub>  | Data Hold After WR                 | 120       |      | ns   | C <sub>L</sub> = 20pF |
| t <sub>CY</sub>  | Cycle Time                         | 2.5       | 15.0 | μs   | 6 MHz XTAL = 2.5      |
| t <sub>DR</sub>  | Data Hold                          | 0         | 200  | ns   |                       |
| t <sub>RD</sub>  | PSEN, RD to Data In                |           | 500  | ns   |                       |
| t <sub>AW</sub>  | Address Setup to WR                | 230       |      | ns   |                       |
| t <sub>AD</sub>  | Address Setup to Data In           |           | 950  | ns   |                       |
| t <sub>AFC</sub> | Address Float to RD, PSEN          | 0         |      | ns   |                       |
| t <sub>CA</sub>  | Control Pulse to ALE               | 10        |      | ns   |                       |

**NOTE 1:** Control outputs: C<sub>L</sub> = 80 pF  
 BUS Outputs: C<sub>L</sub> = 150 pF  
 t<sub>CY</sub> = 2.5 μs for standard parts

# 8243 MCS<sup>®</sup>-48 INPUT/OUTPUT EXPANDER EXPRESS

- 0°C to 70°C Operation
- -40°C to 85°C Operation
- 168 Hr. Burn-in

The new Intel EXPRESS 8243 family is a process enhanced version of the familiar MCS-48 input/output expander. The EXPRESS versions are designed to meet the needs of those applications whose operating requirements exceed commercial standards, but fall short of military conditions.

The EXPRESS options include the commercial standard and -40°C to 85°C operation with or without 168 ± 8 hours of dynamic burn-in a 125°C per MIL-STD-883B, method 1015. Figure 3 summarizes the option marking designators and package selections.

All EXPRESS features and operating characteristics are identical to the standard, commercial grade part except the VCC supply current. This specification is increased by 5mA on -40°C to 85°C EXPRESS options only: 15mA typical/25 mA maximum.

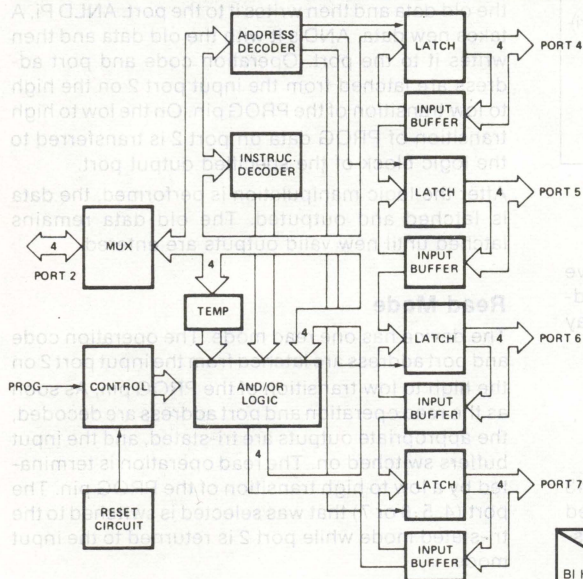


Figure 1. 8243 Block Diagram

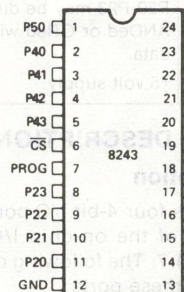


Figure 2. 8243 Pin Configuration

| Temp. Range<br>BI Hours | 0-70   |        | -40-85 |     |
|-------------------------|--------|--------|--------|-----|
|                         | 0      | 0      | 168    | 168 |
| * P8243                 | TP8243 | QP8243 |        |     |
| * D8243                 | TD8243 | QD8243 | LD8243 |     |

\*Commercial grade  
P = Plastic package  
D = Cerdip package

Figure 3. Available 8243 EXPRESS OPTIONS



Table 1. Pin Description

| Symbol          | Pin No.  | Function   |
|-----------------|----------|--|
| PROG            | 7        | Clock Input. A high to low transition on PROG signifies that address and control are available on P20-P23, and a low to high transition signifies that data is available on P20-P23.   |
| $\overline{CS}$ | 6        | Chip Select Input. A high on CS inhibits any change of output or internal status.  |
| P20-P23         | 11-8     | Four (4) bit bi-directional port contains the address and control bits on a high to low transition of PROG. During a low to high transition contains the data for a selected output port if a write operation, or the data from a selected port before the low to high transition if a read operation. |
| GND             | 12       | 0 volt supply.   |
| P40-P43         | 2-5      | Four (4) bit bi-directional I/O ports.   |
| P50-P53         | 1, 23-21 | May be programmed to be input (during read), low impedance   |
| P60-P63         | 20-17    | latched output (after write), or a tri-state (after read). Data on pins  |
| P70-P73         | 13-16    | P20-P23 may be directly written, ANDed or ORed with previous data.   |
| VCC             | 24       | +5 volt supply.  |

## FUNCTIONAL DESCRIPTION

### General Operation

The 8243 contains four 4-bit I/O ports which serve as an extension of the on-chip I/O and are addressed as ports 4-7. The following operations may be performed on these ports:

- Transfer Accumulator to Port.
- Transfer Port to Accumulator.
- AND Accumulator to Port.
- OR Accumulator to Port.

All communication between the 8048 and the 8243 occurs over Port 2 (P20-P23) with timing provided by an output pulse on the PROG pin of the processor. Each transfer consists of two 4-bit nibbles:

The first containing the "op code" and port address and the second containing the actual 4-bits of data. A high to low transition of the PROG line indicates that address is present while a low to high transition indicates the presence of data. Additional 8243's may be added to the 4-bit bus and chip selected using additional output lines from the 8048/8748/8035.

### Power On Initialization

Initial application of power to the device forces input/output ports 4, 5, 6, and 7 to the tri-state and port 2 to the input mode. The PROG pin may be either high or low when power is applied. The first high to low transition of PROG causes device to exit power on mode. The power on sequence is initiated if VCC drops below 1V.

| Address |     | Instruction |       |
|---------|-----|-------------|-------|
| P21     | P20 | Code        | Code  |
| 0       | 0   | Port 4      | 0     |
| 0       | 1   | Port 5      | 0     |
| 1       | 0   | Port 6      | 1     |
| 1       | 1   | Port 7      | 1     |
|         |     |             | Read  |
|         |     |             | Write |
|         |     |             | ORLD  |
|         |     |             | ANLD  |

### Write Modes

The device has three write modes. MOVD Pi, A directly writes new data into the selected port and old data is lost. ORLD Pi, A takes new data, OR's it with the old data and then writes it to the port. ANLD Pi, A takes new data, AND's it with the old data and then writes it to the port. Operation code and port address are latched from the input port 2 on the high to low transition of the PROG pin. On the low to high transition of PROG data on port 2 is transferred to the logic block of the specified output port.

After the logic manipulation is performed, the data is latched and outputted. The old data remains latched until new valid outputs are entered.

### Read Mode

The device has one read mode. The operation code and port address are latched from the input port 2 on the high to low transition of the PROG pin. As soon as the read operation and port address are decoded, the appropriate outputs are tri-stated, and the input buffers switched on. The read operation is terminated by a low to high transition of the PROG pin. The port (4, 5, 6 or 7) that was selected is switched to the tri-stated mode while port 2 is returned to the input mode.

Normally, a port will be in an output (write mode) or input (read mode). If modes are changed during operation, the first read following a write should be ignored; all following reads are valid. This is to allow the external driver on the port to settle after the first read instruction removes the low impedance drive from the 8243 output. A read of any port will leave that port in a high impedance state.



# ABSOLUTE MAXIMUM RATINGS\*

Ambient Temperature Under Bias ..... 0°C to 70°C  
 Storage Temperature ..... -65°C to +150°C  
 Voltage on Any Pin  
   With Respect to Ground ..... -0.5 V to +7V  
 Power Dissipation ..... 1 Watt

\*NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

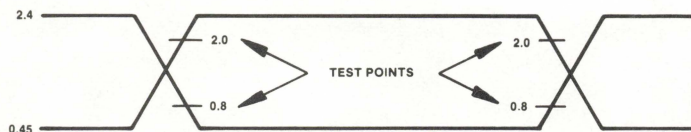
# D.C. CHARACTERISTICS $T_A = 0^\circ\text{C to } 70^\circ\text{C}$ , $V_{CC} = 5\text{V } 10\%$

| Symbol | Parameter                      | Min. | Typ. | Max.         | Units   | Test Conditions   |
|--------|--------------------------------|------|------|--------------|---------|-------------------|
| VIL    | Input Low Voltage              | -0.5 |      | 0.8          | V       |                   |
| VIH    | Input High Voltage             | 2.0  |      | $V_{CC}+0.5$ | V       |                   |
| VOL1   | Output Low Voltage Ports 4-7   |      |      | 0.45         | V       | IOL = 4.5 mA*     |
| VOL2   | Output Low Voltage Port 7      |      |      | 1            | V       | IOL = 20 mA       |
| VOH1   | Output High Voltage Ports 4-7  | 2.4  |      |              | V       | IOH = 240 $\mu$ A |
| IIL1   | Input Leakage Ports 4-7        | -10  |      | 20           | $\mu$ A | Vin = VCC to OV   |
| IIL2   | Input Leakage Port 2, CS, PROG | -10  |      | 10           | $\mu$ A | Vin = VCC to OV   |
| VOL3   | Output Low Voltage Port 2      |      |      | .45          | V       | IOL = 0.6 mA      |
| ICC    | VCC Supply Current             |      | 10   | 20           | mA      | NOTE 1            |
| VOH2   | Output Voltage Port 2          | 2.4  |      |              |         | IOH = 100 $\mu$ A |
| IOL    | Sum of all IOL from 16 Outputs |      |      | 72           | mA      | 4.5 mA Each Pin   |

\*See following graph for additional sink current capability

# A.C. CHARACTERISTICS $T_A = 0^\circ\text{C to } 70^\circ\text{C}$ , $V_{CC} = 5\text{V } 10\%$

| Symbol | Parameter                               | Min. | Max. | Units | Test Conditions |
|--------|---|------|------|-------|-----------------|
| tA     | Code Valid Before PROG                  | 100  |      | ns    | 80 pF Load      |
| tB     | Code Valid After PROG                   | 60   |      | ns    | 20 pF Load      |
| tC     | Data Valid Before PROG                  | 200  |      | ns    | 80 pF Load      |
| tD     | Data Valid After PROG                   | 20   |      | ns    | 20 pF Load      |
| tH     | Floating After PROG                     | 0    | 150  | ns    | 20 pF Load      |
| tK     | PROG Negative Pulse Width               | 700  |      | ns    |                 |
| tCS    | CS Valid Before/After PROG              | 50   |      | ns    |                 |
| tPO    | Ports 4-7 Valid After <sup>3</sup> PROG |      | 700  | ns    | 100 pF Load     |
| tLP1   | Ports 4-7 Valid Before/After PROG       | 100  |      | ns    |                 |
| tACC   | Port 2 Valid After PROG                 |      | 650  | ns    | 80 pF Load      |

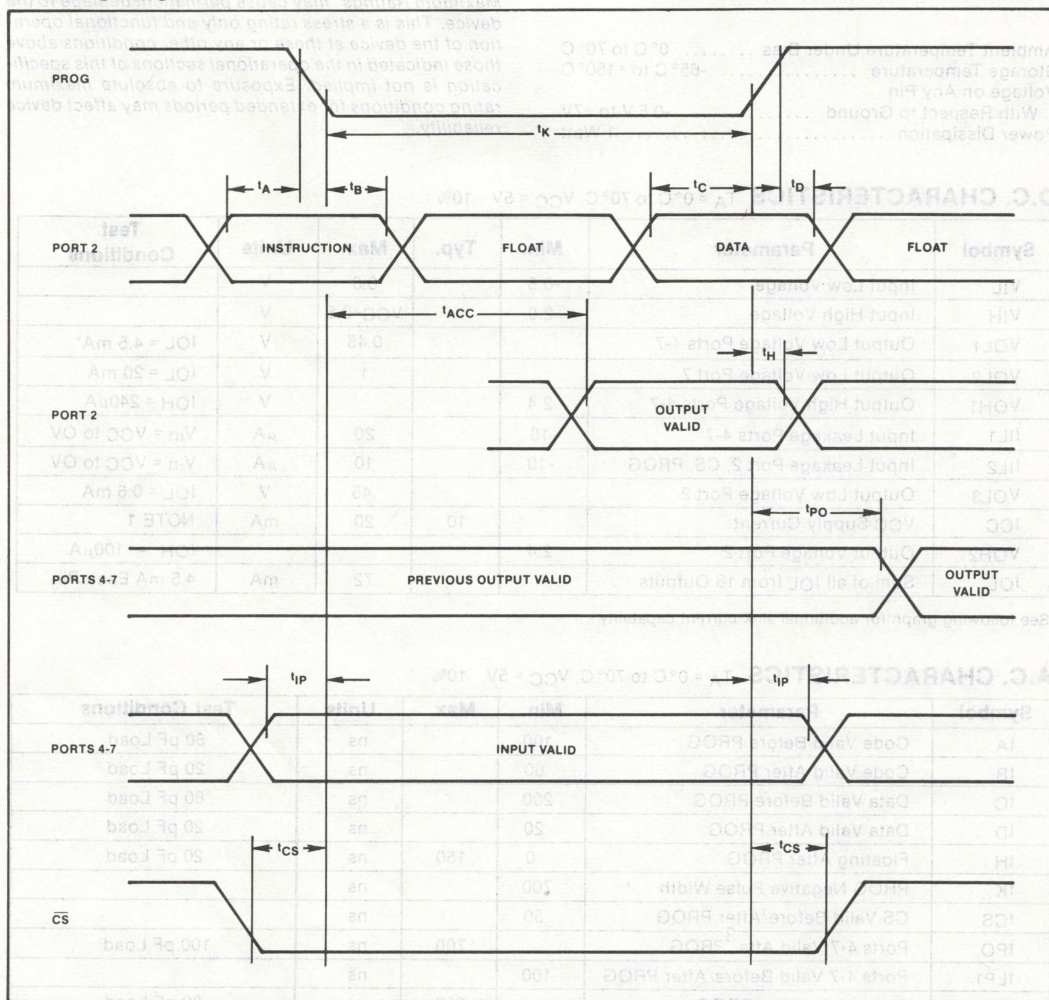


AC Testing: Inputs are driven at 2.4V for a logic "1" and 0.45V for a logic "0". Output timing measurements are made at 2.0V for a logic "1" and 0.8V for a logic "0".

Note 1: ICC (-40° to 85°C EXPRESS options) 15mA typical/25mA maximum.



# WAVEFORMS



AC Testing: Inputs are driven to 5 V for a logic "1" and 0.5 V for a logic "0". Output timing measurements are made at 5.0 V for a logic "1" and 0.5 V for a logic "0".

Note 1: I<sub>CC</sub> (-10° to 85°C EXPRESS options) 15 mA typical/25 mA maximum.

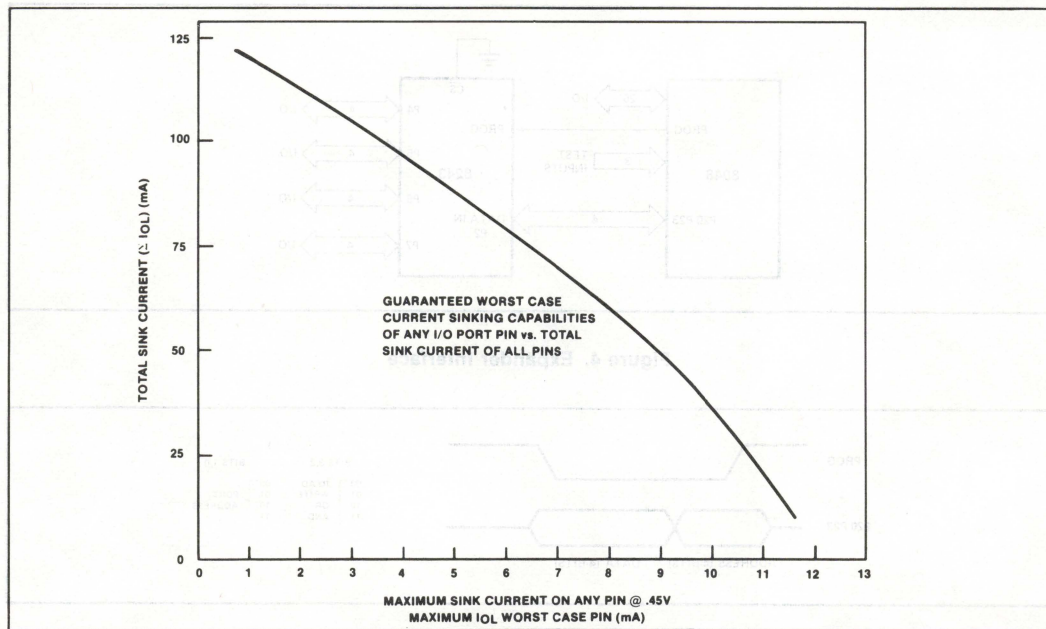


Figure 3

## Sink Capability

The 8243 can sink 5 mA @ .45V on each of its 16 I/O lines simultaneously. If, however, all lines are not sinking simultaneously or all lines are not fully loaded, the drive capability of any individual line increases as is shown by the accompanying curve.

For example, if only 5 of the 16 lines are to sink current at one time, the curve shows that each of those 5 lines is capable of sinking 9 mA @ .45V (if any lines are to sink 9 mA the total IOL must not exceed 45 mA or five 9 mA loads).

Example: How many pins can drive 5 TTL loads (1.6 mA) assuming remaining pins are unloaded?

$$\begin{aligned} IOL &= 5 \times 1.6 \text{ mA} = 8 \text{ mA} \\ \epsilon IOL &= 60 \text{ mA from curve} \\ \# \text{ pins} &= 60 \text{ mA} \div 8 \text{ mA/pin} = 7.5 = 7 \end{aligned}$$

In this case, 7 lines can sink 8 mA for a total of 56mA. This leaves 4 mA sink current capability which can be divided in any way among the remaining 8 I/O lines of the 8243.

Example: This example shows how the use of the 20 mA sink capability of Port 7 affects the sinking capability of the other I/O lines.

An 8243 will drive the following loads simultaneously.

- 2 loads—20 mA @ 1V (port 7 only)
- 8 loads—4 mA @ .45V
- 6 loads—3.2 mA @ .45V
- Is this within the specified limits?

$$\begin{aligned} \epsilon IOL &= (2 \times 20) + (8 \times 4) + (6 \times 3.2) = 91.2 \text{ mA} \\ \text{From the curve: for } IOL &= 4 \text{ mA, } \epsilon IOL \approx 93 \text{ mA} \\ \text{Since } 91.2 \text{ mA} &< 93 \text{ mA the loads are within} \\ &\text{specified limits.} \end{aligned}$$

Although the 20 mA @ 1V loads are used in calculating  $\epsilon IOL$ , it is the largest current required @ .45V which determines the maximum allowable  $\epsilon IOL$ .

**NOTE:** A 10 to 50K  $\Omega$  pullup resistor to +5V should be added to 8243 outputs when driving to 5V CMOS directly.



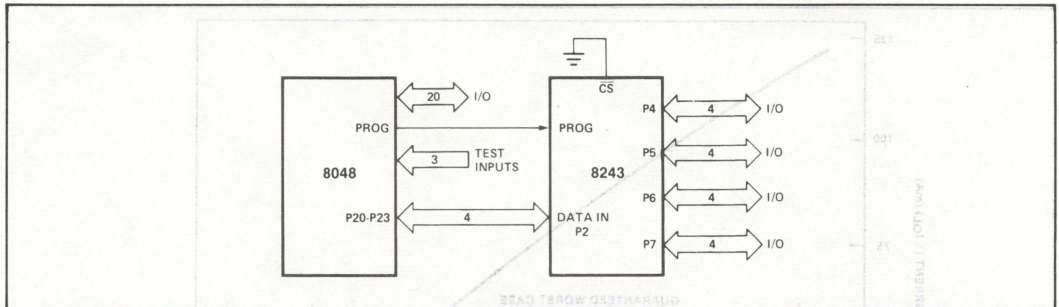


Figure 4. Expander Interface

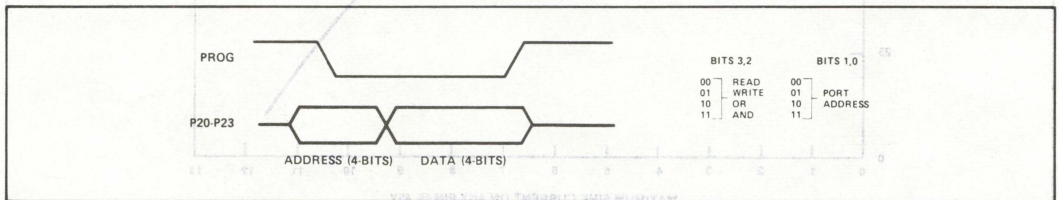


Figure 5. Output Expander Timing

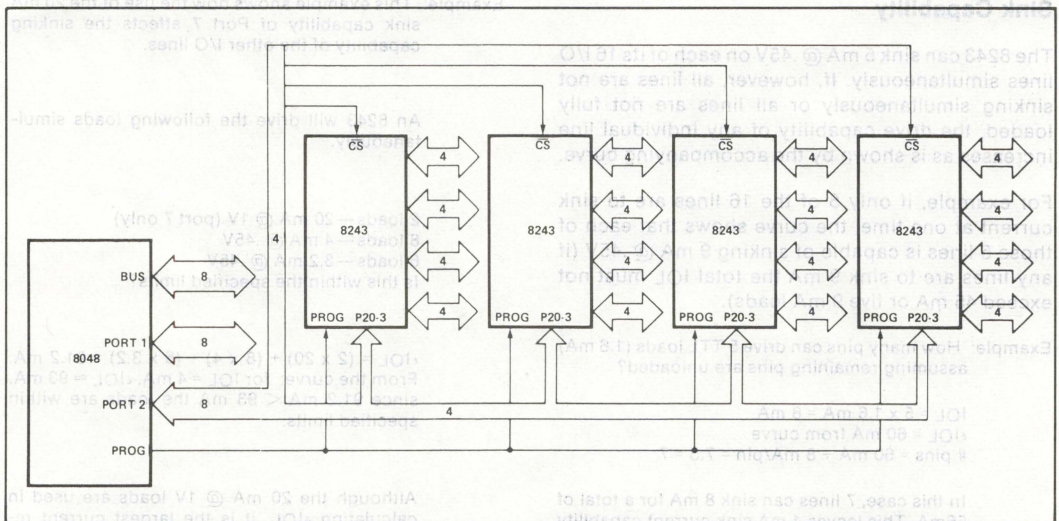


Figure 6. Using Multiple 8243's

# SINGLE-COMPONENT 8-BIT MICROCOMPUTERS EXPRESS

■ 0°C to 70°C Operation

■ -40°C to 85°C Operation

■ 168 Hr Burn-in

| Symbol          | Parameter                                    | Limit           | Unit | Test Conditions |
|-----------------|--|-----------------|------|-----------------|
| V <sub>ih</sub> | Input High Voltage (All except XTAL1, XTAL2) | V <sub>CC</sub> | V    |                 |
| I <sub>DD</sub> | V <sub>DD</sub> Supply Current               | 12              | mA   |                 |
| I <sub>CC</sub> | Total Supply Current                         | 98              | mA   |                 |
| V <sub>DD</sub> | RAM Standby Pin Voltage                      | 3.0             | V    | RESET ≤ 0.6V    |

The new Intel EXPRESS family of single-component 8-bit microcomputers offers enhanced processing options to the familiar 8048H/8035HL, 8748, 8049H/8039HL and 8022 Intel components. These EXPRESS products are designed to meet the needs of those applications whose operating requirements exceed commercial standards, but fall short of military conditions.

The EXPRESS options include the commercial standard and -40° to 85°C operation with or without 168±8 hours of dynamic burn-in at 125°C per MIL-STD-883B, method 1015. Figure 1 summarizes the option marking designators and package selections.

For a complete description of 8048H/8035HL, 8748, 8049H/8039HL, and 8022 features and operating characteristics, refer to the respective standard commercial grade data sheet. This document highlights only the electrical specifications which differ from the respective commercial part.

| BI<br>Hrs. | Temp.<br>Range<br>0°C | * 0-70  |          | -40-85   |                     |
|------------|-----------------------|---------|----------|----------|---------------------|
|            |                       | 0       | 0        | 168      | 168                 |
|            |                       | P8048H  | TP8048H  | QP8048H  | <del>LD8048H</del>  |
|            |                       | D8048H  | TD8048H  | QD8048H  | <del>LD8048H</del>  |
|            |                       | P8035HL | TP8035HL | QP8035HL | <del>LD8035HL</del> |
|            |                       | D8035HL | TD8035HL | QD8035HL | <del>LD8035HL</del> |
|            |                       | D8748   | TD8748   | QD8748   | <del>LD8748</del>   |
|            |                       | P8049H  | TP8049H  | QP8049H  | <del>LD8049H</del>  |
|            |                       | D8049H  | TD8049H  | QD8049H  | <del>LD8049H</del>  |
|            |                       | P8039HL | TP8039HL | QP8039HL | <del>LD8039HL</del> |
|            |                       | D8039HL | TD8039HL | QD8039HL | <del>LD8039HL</del> |
|            |                       | P8022   | TP8022   | QP8022   | <del>LD8022</del>   |
|            |                       | D8022   | TD8022   | QD8022   | <del>LD8022</del>   |
|            |                       | P8243   | TP8243   | QP8243   | <del>LD8243</del>   |
|            |                       | D8243   | TD8243   | QD8243   | <del>LD8243</del>   |

\* Commercial grade

P=plastic package

D=cerdip package

Figure 1. EXPRESS Options



Extended Temperature Electrical Specification Deviations\*

8048H/8035HL

DC CHARACTERISTICS ( $T_A = -40^\circ\text{C}$  to  $85^\circ\text{C}$ ;  $V_{CC} = V_{DD} = 5V \pm 10\%$ ;  $V_{SS} = 0V$ )

| Symbol            | Parameter   | Limits |     |          | Unit | Test Conditions   |
|-------------------|---|--------|-----|----------|------|-------------------|
|                   |   | Min    | Typ | Max      |      |                   |
| $V_{IH}$          | Input High Voltage (All except XTAL1, XTAL2, RESET) | 2.2    |     | $V_{CC}$ | V    |                   |
| $I_{DD}$          | $V_{DD}$ Supply Current                             |        | 6   | 12       | mA   |                   |
| $I_{DD} + I_{CC}$ | Total Supply Current                                |        | 50  | 95       | mA   |                   |
| $V_{DD}$          | RAM Standby Pin Voltage                             | 3.0    |     |          | V    | RESET $\leq 0.6V$ |

8049H/8039HL

DC CHARACTERISTICS ( $T_A = -40^\circ\text{C}$  to  $85^\circ\text{C}$ ;  $V_{CC} = V_{DD} = 5V \pm 10\%$ ;  $V_{SS} = 0V$ )

| Symbol            | Parameter   | Limits |     |          | Unit | Test Conditions |
|-------------------|---|--------|-----|----------|------|-----------------|
|                   |   | Min    | Typ | Max      |      |                 |
| $V_{IH}$          | Input High Voltage (All except XTAL1, XTAL2, RESET) | 2.2    |     | $V_{CC}$ | V    |                 |
| $I_{DD}$          | $V_{DD}$ Supply Current                             |        | 10  | 20       | mA   |                 |
| $I_{DD} + I_{CC}$ | Total Supply Current                                |        | 75  | 120      | mA   |                 |

8748

DC CHARACTERISTICS ( $T_A = -40^\circ\text{C}$  to  $85^\circ\text{C}$ ;  $V_{CC} = V_{DD} = 5V \pm 10\%$ ;  $V_{SS} = 0V$ )

| Symbol            | Parameter   | Limits |     |          | Unit    | Test Conditions                        |
|-------------------|---|--------|-----|----------|---------|--|
|                   |   | Min    | Typ | Max      |         |  |
| $V_{IH}$          | Input High Voltage (All except XTAL1, XTAL2, RESET)     | 2.2    |     | $V_{CC}$ | V       |  |
| $I_{LI1}$         | Input Leakage Current (P10-P17, P20-27, EA $\bar{SS}$ ) |        |     | -600     | $\mu A$ | $V_{SS} + .45 \leq V_{IN} \leq V_{CC}$ |
| $I_{DD}$          | $V_{DD}$ Supply Current                                 |        | 10  | 20       | mA      |  |
| $I_{DD} + I_{CC}$ | Total Supply Current                                    |        | 75  | 145      | mA      |  |

\*Refer to individual commercial grade data sheets for complete operating characteristics.



## Extended Temperature Electrical Specification Deviations\*

8748

AC CHARACTERISTICS ( $T_A = -40^\circ\text{C}$  to  $85^\circ\text{C}$ ;  $V_{CC} = V_{DD} = 5\text{V} \pm 10\%$ ;  $V_{SS} = 0\text{V}$ )

| Symbol | Parameter                                      | Min  | Max  | Unit          | Test Conditions      |
|--------|--|------|------|---------------|----------------------|
| tCP    | Port Control Setup Before Falling Edge of PROG | 115  |      | ns            |                      |
| tPC    | Port Control Hold After Falling Edge of PROG   | 65   |      | ns            |                      |
| tPR    | PROG to Time P2 Input Must Be Valid            |      | 860  | ns            |                      |
| tPF    | Input Data Hold Time                           | 0    | 160  | ns            |                      |
| tDP    | Output Data Setup Time                         | 230  |      | ns            |                      |
| tPD    | Output Data Hold Time                          | 25   |      | ns            |                      |
| tPP    | PROG Pulse Width                               | 920  |      | ns            |                      |
| tPL    | Port 2 I/O Data Setup                          | 300  |      | ns            |                      |
| tLP    | Port 2 I/O Data Hold                           | 120  |      | ns            |                      |
| tLL    | ALE Pulse Width                                | 300  |      | ns            |                      |
| tAL    | Address Setup to ALE                           | 120  |      | ns            |                      |
| tLA    | Address Hold from ALE                          | 80   |      | ns            |                      |
| tCC    | Control Pulse Width (PSEN, RD, WR)             | 600  |      | ns            |                      |
| tDW    | Data Setup Before WR                           | 600  |      | ns            |                      |
| tWD    | Data Hold After WR                             | 120  |      | ns            | $C_L = 20\text{ pF}$ |
| tCY    | Cycle Time                                     | 4.17 | 15.0 | $\mu\text{s}$ | (3.6 MHz XTAL)       |
| tDR    | Data Hold                                      | 0    | 200  | ns            |                      |
| tRD    | PSEN, RD to Data In                            |      | 600  | ns            |                      |
| tAW    | Address Setup to WR                            | 260  |      | ns            |                      |
| tAD    | Address Setup to Data In                       |      | 900  | ns            |                      |
| tAFC   | Address Float to RD, PSEN                      | -60  |      | ns            |                      |
| tCA    | Control Pulse to ALE                           | 10   |      | ns            |                      |

Note: Control Outputs:  $C_L = 80\text{ pF}$ ; Bus Outputs:  $C_L = 150\text{ pF}$ ;  $t_{CY} = 4.17\text{ }\mu\text{sec}$ 

\*Refer to individual commercial grade data sheets for complete operating characteristics.



### Extended Temperature Electrical Specification Deviations\*

## 8022

**DC CHARACTERISTICS** ( $T_A = -40^{\circ}\text{C}$  to  $85^{\circ}\text{C}$ ;  $V_{CC} = 5.5\text{V} \pm 1\text{V}$ ;  $V_{SS} = 0\text{V}$ )

| Symbol    | Parameter   | Limits    |     | Unit      | Test Conditions  |
|-----------|---|-----------|-----|-----------|--|
|           |   | Min       | Typ |           |  |
| $V_{IL1}$ | Input Low Voltage (Port 0)                                | -0.5      |     | VTH -0.2  | V  |
| $V_{IH}$  | Input High Voltage<br>(All except XTAL1, RESET)           | 2.3       |     | $V_{CC}$  | V<br>$V_{CC} = 5.0\text{V}$<br>$\pm 10\%$ VTH Floating     |
| $V_{IH1}$ | Input High Voltage<br>(All except XTAL1, RESET)           | 3.8       |     | $V_{CC}$  | V<br>$V_{CC} = 5.5\text{V} \pm 1\text{V}$<br>VTH Floating  |
| $V_{IH2}$ | Input High Voltage (Port 0)                               | VTH + 0.2 |     | $V_{CC}$  | V  |
| $V_{IH3}$ | Input High Voltage (RESET, XTAL1)                         | 3.8       |     | $V_{CC}$  | V  |
| $V_{IL}$  | Input Low Voltage   | -0.5      |     | 0.5       | V  |
| $V_{OL}$  | Output Low Voltage  |           |     | 0.45      | V<br>IOL = 0.8mA   |
| $V_{OL1}$ | Output Low Voltage (P10, P11)                             |           |     | 2.5       | V<br>IOL = 3mA   |
| $V_{OH}$  | Output High Voltage (All unless open drain option Port 0) | 2.4       |     |           | V<br>IOH = 30 $\mu$ A                                      |
| $I_{LI}$  | Input Current (T1)  |           |     | $\pm 700$ | $\mu$ A<br>$V_{CC} \geq V_{IN} \geq V_{SS} + 0.45\text{V}$ |
| $I_{LI1}$ | Input Current to Ports                                    |           |     | 500       | $\mu$ A<br>$V_{IN} = 0.45\text{V}$                         |
| $I_{CC}$  | $V_{CC}$ Supply Current                                   |           |     | 120       | mA   |

**AC CHARACTERISTICS** ( $T_A = -40^{\circ}\text{C}$  to  $85^{\circ}\text{C}$ ;  $V_{CC} = 5.5\text{V} \pm 1\text{V}$ ;  $V_{SS} = 0\text{V}$ )

| Symbol | Parameter                                | Min  | Max       | Unit    | Test Conditions                  |
|--------|--|------|-----------|---------|----------------------------------|
| tCY    | Cycle Time                               | 8.38 | 50.0      | $\mu$ s | 3.58 MHz XTAL = 8.38 $\mu$ s tCY |
| VT1    | Zero-Cross Detection Input (T1)          | 1    | 3         | VACpp   | AC Coupled                       |
| AZC    | Zero-Cross Accuracy                      |      | $\pm 200$ | mV      | 60Hz Sine Wave                   |
| FT1    | Zero-Cross Detection Input Frequency(T1) | 0.05 | 1         | kHz     |                                  |
| tLL    | ALE Pulse Width                          | 3.9  | 23.0      | $\mu$ s | tCY = 8.38 $\mu$ s for min       |

Note: Control Outputs:  $C_L = 80$  pf; tCY = 8.38  $\mu$ sec.

**A/D CONVERTER CHARACTERISTICS** ( $AV_{CC} = 5.5\text{V} \pm 1\text{V}$ ;  $AV_{SS} = 0\text{V}$ ;  $AV_{CC}/2 \leq V_{AREF} \leq AV_{CC}$ )

| Parameter         | Limits |     |                           | Unit | Test Conditions |
|-------------------|--------|-----|---------------------------|------|-----------------|
|                   | Min    | Typ | Max                       |      |                 |
| Absolute Accuracy |        |     | 1.6% FSR<br>$\pm 1/2$ LSB | LSB  |                 |

Note: The analog input must be maintained at a constant voltage during the sample time (tss + tsh).

\*Refer to individual commercial grade data sheets for complete operating characteristics.



# M8048/M8748/M8035L SINGLE COMPONENT 8-BIT MICROCOMPUTER

MILITARY

- 8048 Mask Programmable ROM
- 8748 User Programmable/Erased EPROM
- 8035L Requires External ROM or EPROM

- -55°C to +125°C 6 MHz Operation (M8048/M8035L)
- -55°C to +125°C 3.6 MHz Operation (M8748)
- 8-Bit CPU, ROM, RAM, I/O in Single Package
- Interchangeable ROM and EPROM Versions
- Single 5V Supply
- 2.5  $\mu$ sec and 5.0  $\mu$ sec Cycle Versions
- All Instructions 1 or 2 Cycles.
- Over 90 Instructions: 70% Single Byte
- 1K x 8 ROM/EPROM
- 64 x 8 RAM
- 27 I/O Lines
- Interval Timer/Event Counter
- Easily Expandable Memory and I/O
- Compatible with 8080/8085 Series Peripherals
- Single Level Interrupt
- Screened to MIL-STD-883B

The Intel M8048/M8748/M8035L are totally self-sufficient 8-bit parallel computers fabricated on single silicon chips using Intel's N-Channel silicon gate MOS process.

The M8048 contains an 8-bit CPU, a 1K x 8 program memory, a 64 x 8 RAM data memory, 27 I/O lines, and an 8-bit timer/counter in addition to on-board oscillator and clock circuits. For systems that require extra capability, the M8048 can be expanded using standard memories and MCS-80\*/MCS-85\* peripherals. The M8035L is the equivalent of an M8048 without program memory, and has the RAM power down mode of the M8048. To reduce development problems to a minimum and provide maximum flexibility, three interchangeable pin-compatible\* versions of this single component micro-computer exist: the M8748 with user-programmable and erasable EPROM program memory for prototype and preproduction systems, the M8048 with factory-programmed mask ROM program memory for low cost, high volume production, and the M8035L without program memory for use with external program memories.

This microprocessor is designed to be an efficient controller as well as an arithmetic processor. The M8048 has extensive bit handling capability as well as facilities for both binary and BCD arithmetic. Efficient use of program memory results from an instruction set consisting mostly of single byte instructions and no instructions over 2 bytes in length.

\*V<sub>DD</sub> is used to program the M8748 and used for low power standby on the M8048/8035L.

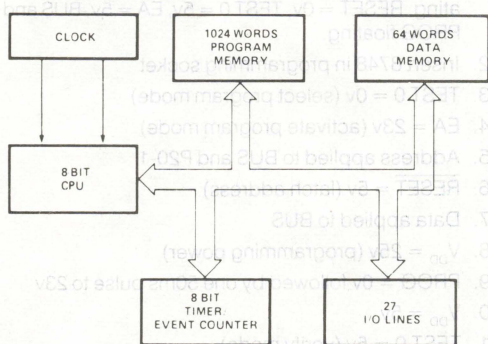


Figure 1. Block Diagram

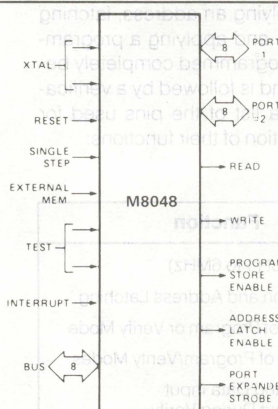


Figure 2. Logic Symbol

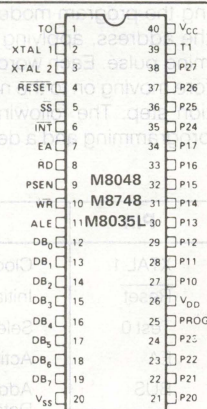


Figure 3. Pin Configuration



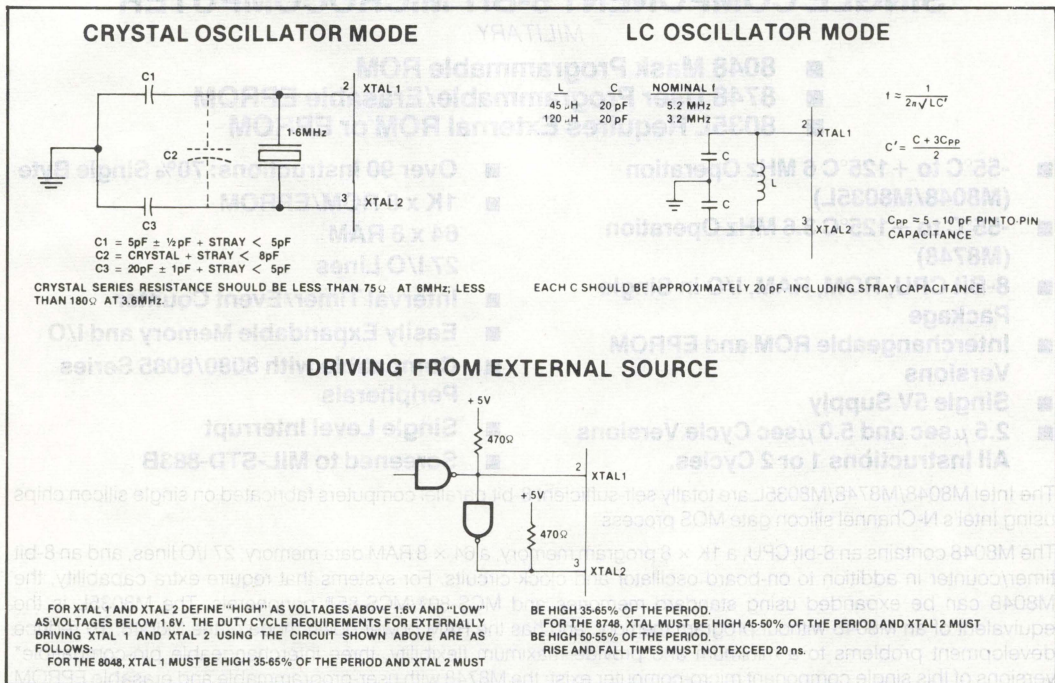


Figure 4

## PROGRAMMING, VERIFYING, AND ERASING THE 8748 EPROM

### Programming Verification

In brief, the programming process consists of: activating the program mode, applying an address, latching the address, applying data, and applying a programming pulse. Each word is programmed completely before moving on to the next and is followed by a verification step. The following is a list of the pins used for programming and a description of their functions:

| Pin             | Function  |
|-----------------|---|
| XTAL 1          | Clock Input (1 to 6MHz)                             |
| Reset           | Initialization and Address Latching                 |
| Test 0          | Selection of Program or Verify Mode                 |
| EA              | Activation of Program/Verify Modes                  |
| BUS             | Address and Data Input<br>Data Output During Verify |
| P20-1           | Address Input                                       |
| V <sub>DD</sub> | Programming Power Supply                            |
| PROG            | Program Pulse Input                                 |

#### WARNING:

An attempt to program a missocketed 8748 will result in severe damage to the part. An indication of a properly socketed part is the appearance of the ALE clock output. The lack of this clock may be used to disable the programmer.

The Program/Verify sequence is:

1.  $V_{DD} = 5v$ , Clock applied or internal oscillator operating,  $\overline{RESET} = 0v$ ,  $TEST\ 0 = 5v$ ,  $EA = 5v$ , BUS and PROG floating.
2. Insert 8748 in programming socket
3.  $TEST\ 0 = 0v$  (select program mode)
4.  $EA = 23v$  (activate program mode)
5. Address applied to BUS and P20-1
6.  $\overline{RESET} = 5v$  (latch address)
7. Data applied to BUS
8.  $V_{DD} = 25v$  (programming power)
9.  $PROG = 0v$  followed by one 50ms pulse to 23v
10.  $V_{DD} = 5v$
11.  $TEST\ 0 = 5v$  (verify mode)
12. Read and verify data on BUS
13.  $TEST\ 0 = 0v$
14.  $\overline{RESET} = 0v$  and repeat from step 5
15. Programmer should be at condition of step 1 when 8748 is removed from socket.

# A.C. TIMING SPECIFICATION FOR PROGRAMMING

$T_A = 25^\circ\text{C} \pm 5^\circ\text{C}$ ,  $V_{CC} = 5V \pm 5\%$ ,  $V_{DD} = 25V \pm 1V$

| Symbol     | Parameter                                  | Min.      | Max.      | Unit          | Test Conditions |
|------------|--|-----------|-----------|---------------|-----------------|
| $t_{AW}$   | Address Setup Time to RESET $\uparrow$     | $4t_{CY}$ |           |               |                 |
| $t_{WA}$   | Address Hold Time After RESET $\uparrow$   | $4t_{CY}$ |           |               |                 |
| $t_{DW}$   | Data in Setup Time to PROG $\uparrow$      | $4t_{CY}$ |           |               |                 |
| $t_{WD}$   | Data in Hold Time After PROG $\downarrow$  | $4t_{CY}$ |           |               |                 |
| $t_{PH}$   | RESET Hold Time to Verify                  | $4t_{CY}$ |           |               |                 |
| $t_{VDDW}$ | $V_{DD}$                                   | $4t_{CY}$ |           |               |                 |
| $t_{VDDH}$ | $V_{DD}$ Hold Time After PROG $\downarrow$ | 0         |           |               |                 |
| $t_{PW}$   | Program Pulse Width                        | 50        | 60        | mS            |                 |
| $t_{TW}$   | Test 0 Setup Time for Program Mode         | $4t_{CY}$ |           |               |                 |
| $t_{WT}$   | Test 0 Hold Time After Program Mode        | $4t_{CY}$ |           |               |                 |
| $t_{DO}$   | Test 0 to Data Out Delay                   |           | $4t_{CY}$ |               |                 |
| $t_{WW}$   | RESET Pulse Width to Latch Address         | $4t_{CY}$ |           |               |                 |
| $t_r, t_f$ | $V_{DD}$ and PROG Rise and Fall Times      | 0.5       | 2.0       | $\mu\text{S}$ |                 |
| $t_{CY}$   | CPU Operation Cycle Time                   | 5.0       |           | $\mu\text{S}$ |                 |
| $t_{RE}$   | RESET Setup Time Before EA $\uparrow$      | $4t_{CY}$ |           |               |                 |

Note: If Test 0 is high  $t_{DO}$  can be triggered by RESET  $\uparrow$ .

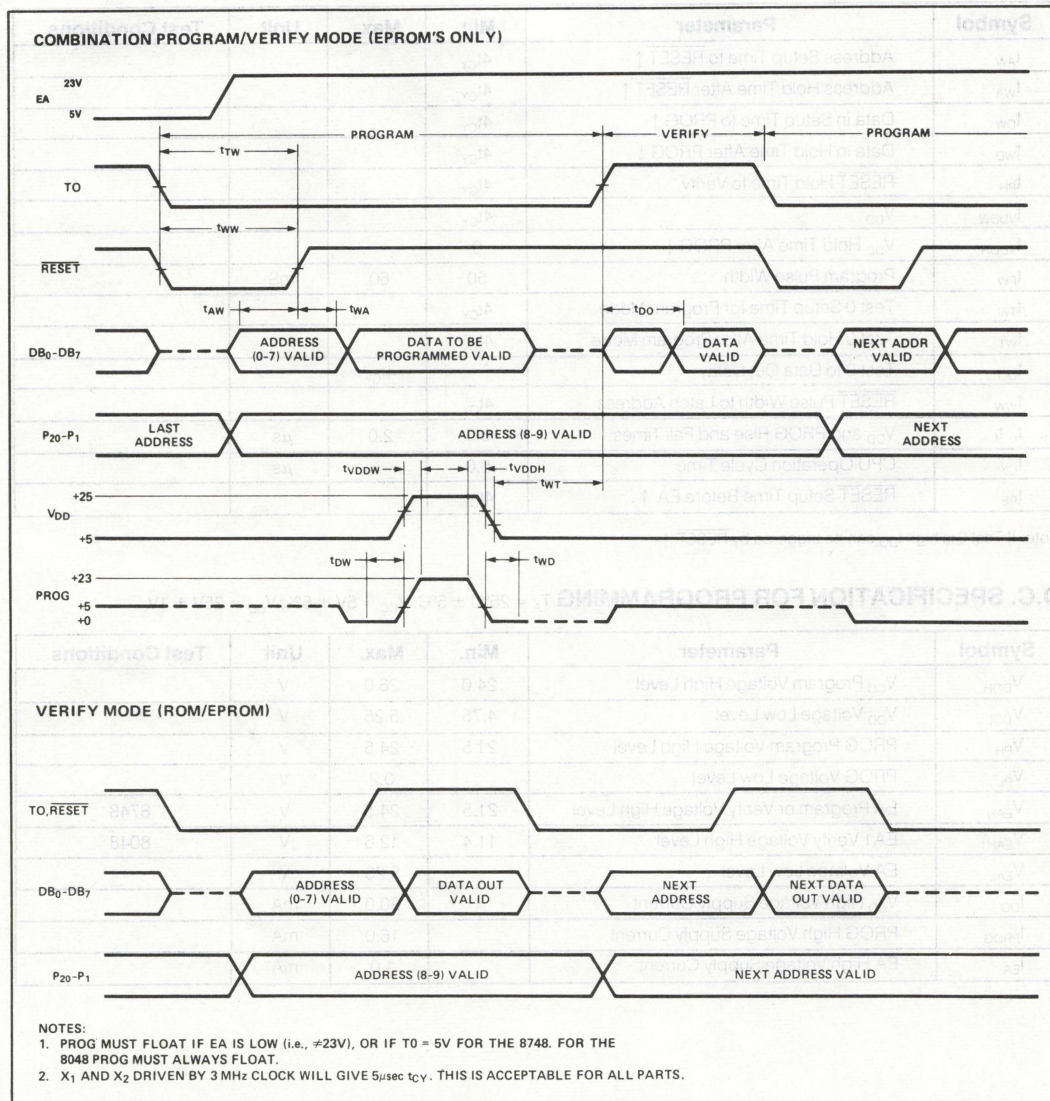
# D.C. SPECIFICATION FOR PROGRAMMING

$T_A = 25^\circ\text{C} \pm 5^\circ\text{C}$ ,  $V_{CC} = 5V \pm 5\%$ ,  $V_{DD} = 25V \pm 1V$

| Symbol     | Parameter                               | Min. | Max. | Unit | Test Conditions |
|------------|---|------|------|------|-----------------|
| $V_{DOH}$  | $V_{DD}$ Program Voltage High Level     | 24.0 | 26.0 | V    |                 |
| $V_{DDL}$  | $V_{DD}$ Voltage Low Level              | 4.75 | 5.25 | V    |                 |
| $V_{PH}$   | PROG Program Voltage High Level         | 21.5 | 24.5 | V    |                 |
| $V_{PL}$   | PROG Voltage Low Level                  |      | 0.2  | V    |                 |
| $V_{EAH}$  | EA Program or Verify Voltage High Level | 21.5 | 24.5 | V    | 8748            |
| $V_{EAH1}$ | EA1 Verify Voltage High Level           | 11.4 | 12.6 | V    | 8048            |
| $V_{EAL}$  | EA Voltage Low Level                    |      | 5.25 | V    |                 |
| $I_{DD}$   | $V_{DD}$ High Voltage Supply Current    |      | 30.0 | mA   |                 |
| $I_{PROG}$ | PROG High Voltage Supply Current        |      | 16.0 | mA   |                 |
| $I_{EA}$   | EA High Voltage Supply Current          |      | 1.0  | mA   |                 |



# WAVEFORMS



The 8748 EPROM can be programmed by either of two Intel products:

1. PROMPT-48 Microcomputer Design Aid, or
2. Universal PROM Programmer (UPP Series) peripheral of the Intellec® Development System with a UPP-848 Personality Card.

Table 1. Instruction Set Summary

|              | Mnemonic       | Description                   | Bytes | Cycle |
|--------------|----------------|-------------------------------|-------|-------|
| Accumulator  | ADD A, R       | Add register to A             | 1     | 1     |
|              | ADD A, @R      | Add data memory to A          | 1     | 1     |
|              | ADD A, #data   | Add immediate to A            | 2     | 2     |
|              | ADDC A, R      | Add register with carry       | 1     | 1     |
|              | ADDC A, @R     | Add data memory with carry    | 1     | 1     |
|              | ADDC A, #data  | Add immediate with carry      | 2     | 2     |
|              | ANL A, R       | And register to A             | 1     | 1     |
|              | ANL A, @R      | And data memory to A          | 1     | 1     |
|              | ANL A, #data   | And immediate to A            | 2     | 2     |
|              | ORL A, R       | Or register to A              | 1     | 1     |
|              | ORL A, @R      | Or data memory to A           | 1     | 1     |
|              | ORL A, #data   | Or immediate to A             | 2     | 2     |
|              | XRL A, R       | Exclusive or register to A    | 1     | 1     |
|              | XRL A, @R      | Exclusive or data memory to A | 1     | 1     |
|              | XRL A, #data   | Exclusive or immediate to A   | 2     | 2     |
|              | INC A          | Increment A                   | 1     | 1     |
|              | DEC A          | Decrement A                   | 1     | 1     |
|              | CLR A          | Clear A                       | 1     | 1     |
|              | CPL A          | Complement A                  | 1     | 1     |
|              | DA A           | Decimal adjust A              | 1     | 1     |
|              | SWAP A         | Swap nibbles of A             | 1     | 1     |
|              | RL A           | Rotate A left                 | 1     | 1     |
|              | RLC A          | Rotate A left through carry   | 1     | 1     |
|              | RR A           | Rotate A right                | 1     | 1     |
|              | RRC A          | Rotate A right through carry  | 1     | 1     |
| Input/Output | IN A, P        | Input port to A               | 1     | 2     |
|              | OUTL P, A      | Output A to port              | 1     | 2     |
|              | ANL P, #data   | And immediate to port         | 2     | 2     |
|              | ORL P, #data   | Or immediate to port          | 2     | 2     |
|              | INS A, BUS     | Input BUS to A                | 1     | 2     |
|              | OUTL BUS, A    | Output A to BUS               | 1     | 2     |
|              | ANL BUS, #data | And immediate to BUS          | 2     | 2     |
|              | ORL BUS, #data | Or immediate to BUS           | 2     | 2     |
|              | MOVD A, P      | Input expander port to A      | 1     | 2     |
| Registers    | MOVD P, A      | Output A to expander port     | 1     | 2     |
|              | ANLD P, A      | And A to expander port        | 1     | 2     |
|              | ORLD P, A      | Or A to expander port         | 1     | 2     |
| Registers    | INC R          | Increment register            | 1     | 1     |
|              | INC @R         | Increment data memory         | 1     | 1     |
|              | DEC R          | Decrement register            | 1     | 1     |
| Branch       | JMP addr       | Jump unconditional            | 2     | 2     |
|              | JMPP @A        | Jump indirect                 | 1     | 2     |
|              | DJNZ R, addr   | Decrement register and skip   | 2     | 2     |
|              | JC addr        | Jump on carry = 1             | 2     | 2     |
|              | JNC addr       | Jump on carry = 0             | 2     | 2     |
|              | JZ addr        | Jump on A zero                | 2     | 2     |
|              | JNZ addr       | Jump on A not zero            | 2     | 2     |
|              | JT0 addr       | Jump on T0 = 1                | 2     | 2     |
|              | JNT0 addr      | Jump on T0 = 0                | 2     | 2     |
|              | JT1 addr       | Jump on T1 = 1                | 2     | 2     |
|              | JNT1 addr      | Jump on T1 = 0                | 2     | 2     |
|              | JF0 addr       | Jump on F0 = 1                | 2     | 2     |
|              | JF1 addr       | Jump on F1 = 1                | 2     | 2     |
|              | JTF addr       | Jump on timer flag            | 2     | 2     |
|              | JNI addr       | Jump on INT = 0               | 2     | 2     |
|              | JBb addr       | Jump on accumulator bit       | 2     | 2     |

|               | Mnemonic      | Description                       | Bytes | Cycles |
|---------------|---------------|-----------------------------------|-------|--------|
| Subroutine    | CALL addr     | Jump to subroutine                | 2     | 2      |
|               | RET           | Return                            | 1     | 2      |
|               | RETR          | Return and restore status         | 1     | 2      |
|               |               |                                   |       |        |
| Flags         | CLR C         | Clear carry                       | 1     | 1      |
|               | CPL C         | Complement carry                  | 1     | 1      |
|               | CLR F0        | Clear flag 0                      | 1     | 1      |
|               | CPL F0        | Complement flag 0                 | 1     | 1      |
|               | CLR F1        | Clear flag 1                      | 1     | 1      |
|               | CPL F1        | Complement flag 1                 | 1     | 1      |
| Data Moves    | MOV A, R      | Move register to A                | 1     | 1      |
|               | MOV A, @R     | Move data memory to a             | 1     | 1      |
|               | MOV A, #data  | Move immediate to A               | 2     | 2      |
|               | MOV R, A      | Move A to register                | 1     | 1      |
|               | MOV @R, A     | Move A to data memory             | 1     | 1      |
|               | MOV R, #data  | Move immediate to register        | 2     | 2      |
|               | MOV @R, #data | Move immediate to data memory     | 2     | 2      |
|               | MOV A, PSW    | Move PSW to A                     | 1     | 1      |
|               | MOV PSW, A    | Move A to PSW                     | 1     | 1      |
|               | XCH A, R      | Exchange A and register           | 1     | 1      |
|               | XCHA, @R      | Exchange A and data memory        | 1     | 1      |
|               | XCHD A, @R    | Exchange nibble of A and register | 1     | 1      |
|               | MOVX A, @R    | Move external data memory to A    | 1     | 2      |
|               | MOVX @R, A    | Move A to external data memory    | 1     | 2      |
|               | MOVP A, @A    | Move to A from current page       | 1     | 2      |
|               | MOVP3 A, @A   | Move to A from page 3             | 1     | 2      |
| Timer/Counter | MOV A, T      | Read timer/counter                | 1     | 1      |
|               | MOV T, A      | Load timer/counter                | 1     | 1      |
|               | STRT T        | Start timer                       | 1     | 1      |
|               | STRT CNT      | Start counter                     | 1     | 1      |
|               | STOP TCNT     | Stop timer/counter                | 1     | 1      |
|               | EN TCNTI      | Enable timer/counter interrupt    | 1     | 1      |
|               | DIS TCNTI     | Disable timer/counter interrupt   | 1     | 1      |
| Control       | EN I          | Enable external interrupt         | 1     | 1      |
|               | DIS I         | Disable external interrupt        | 1     | 1      |
|               | SEL RB0       | Select register bank 0            | 1     | 1      |
|               | SEL RB1       | Select register bank 1            | 1     | 1      |
|               | SEL MB0       | Select memory bank 0              | 1     | 1      |
|               | SEL MB1       | Select memory bank 1              | 1     | 1      |
|               | ENT0 CLK      | Enable clock output on T0         | 1     | 1      |
|               | NOP           | No operation                      | 1     | 1      |

Mnemonics copyright Intel Corporation 1978



Table 2. Pin Description

| Symbol                               | Pin No.        | Function   |
|--------------------------------------|----------------|--|
| V <sub>SS</sub>                      | 20             | Circuit GND potential  |
| V <sub>DD</sub>                      | 26             | Programming power supply; +25V during program, +5V during operation for both ROM and PROM. Low power standby pin in 8048 and 8035L.  |
| V <sub>CC</sub>                      | 40             | Main power supply; +5V during operation and programming.   |
| PROG                                 | 25             | Program pulse (+23V) input pin during 8748 programming.  |
| P10-P17 Port 1                       | 27-34          | 8-bit quasi-bidirectional port.  |
| P20-P27 Port 2                       | 21-24<br>35-38 | 8-bit quasi-bidirectional port. P20-P23 contain the four high order program counter bits during an external program memory fetch and serve as a 4-bit I/O expander bus for 8243.   |
| DB <sub>0</sub> -DB <sub>7</sub> BUS | 12-19          | True bidirectional port which can be written or read synchronously using the RD, WR strobes. The port can also be statically latched.  |
|                                      |                | Contains the 8 low order program counter bits during an external program memory fetch, and receives the addressed instruction under the control of PSEN. Also contains the address and data during an external RAM data store instruction, under control of ALE, RD, and WR. |
| T0                                   | 1              | Input pin testable using the conditional transfer instructions JT0 and JNT0. T0 can be designated as a clock output using ENT0 CLK instruction. T0 is also used during programming.  |
| T1                                   | 39             | Input pin testable using the JT1, and JNT1 instructions. Can be designated the timer/counter input using the STRT CNT instruction.   |

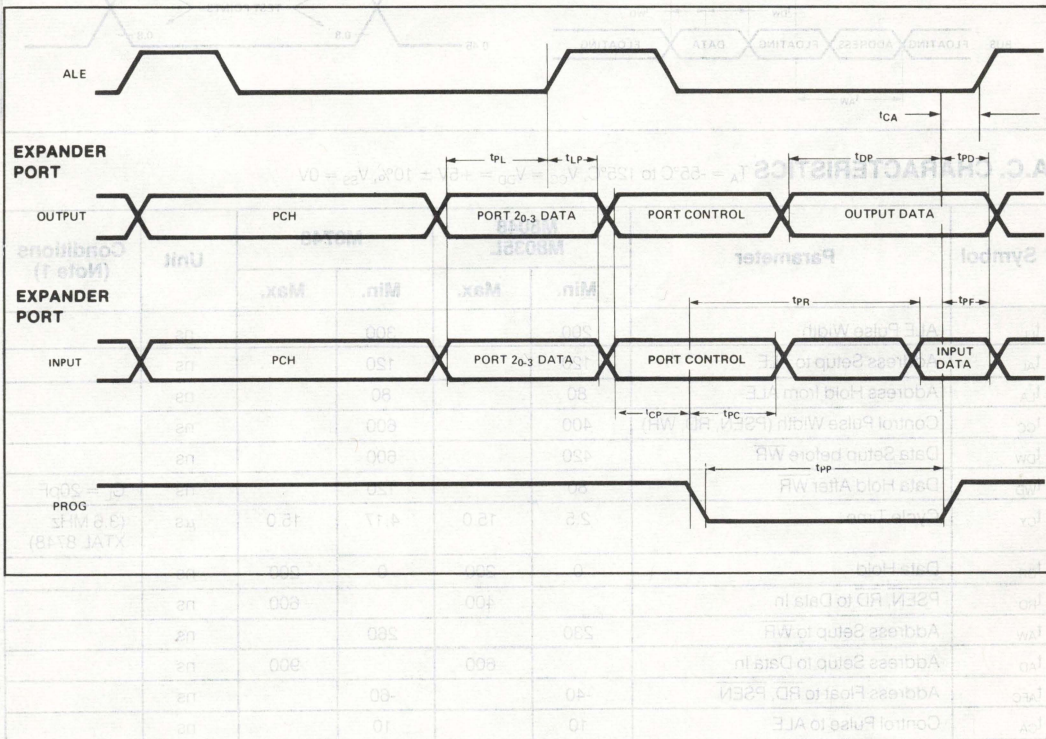
| Symbol | Pin No. | Function  |
|--------|---------|---|
| INT    | 6       | Interrupt input. Initiates an interrupt if interrupt is enabled. Interrupt is disabled after a reset. Also testable with conditional jump instruction. (Active low)                           |
| RD     | 8       | Output strobe activated during a BUS read. Can be used to enable data onto the bus from an external device.<br><br>Used as a read strobe to external data memory. (Active low)                |
| RESET  | 4       | Input which is used to initialize the processor. Also used during PROM programming verification, and power down. (Active low) (Non TTL V <sub>IH</sub> )                                      |
| WR     | 10      | Output strobe during a bus write. (Active low)<br><br>Used as write strobe to external data memory.   |
| ALE    | 11      | Address latch enable. This signal occurs once during each cycle and is useful as a clock output.<br><br>The negative edge of ALE strobes address into external data and program memory.       |
| PSEN   | 9       | Program store enable. This output occurs only during a fetch to external program memory. (Active low)   |
| SS     | 5       | Single step input can be used in junction with ALE to "single step" the processor through each instruction. (Active low)  |
| EA     | 7       | External access input which forces all program memory fetches to reference external memory. Useful for emulation and debug, and essential for testing and program verification. (Active high) |
| XTAL1  | 2       | One side of crystal input for internal oscillator. Also input for external source. (Non TTL V <sub>IH</sub> )   |
| XTAL2  | 3       | Other side of crystal input.  |



# A.C. CHARACTERISTICS (PORT 2 TIMING) $T_A = 55^\circ\text{C}$ to $125^\circ\text{C}$ , $V_{CC} = +5\text{V} \pm 10\%$ , $V_{SS} = 0\text{V}$

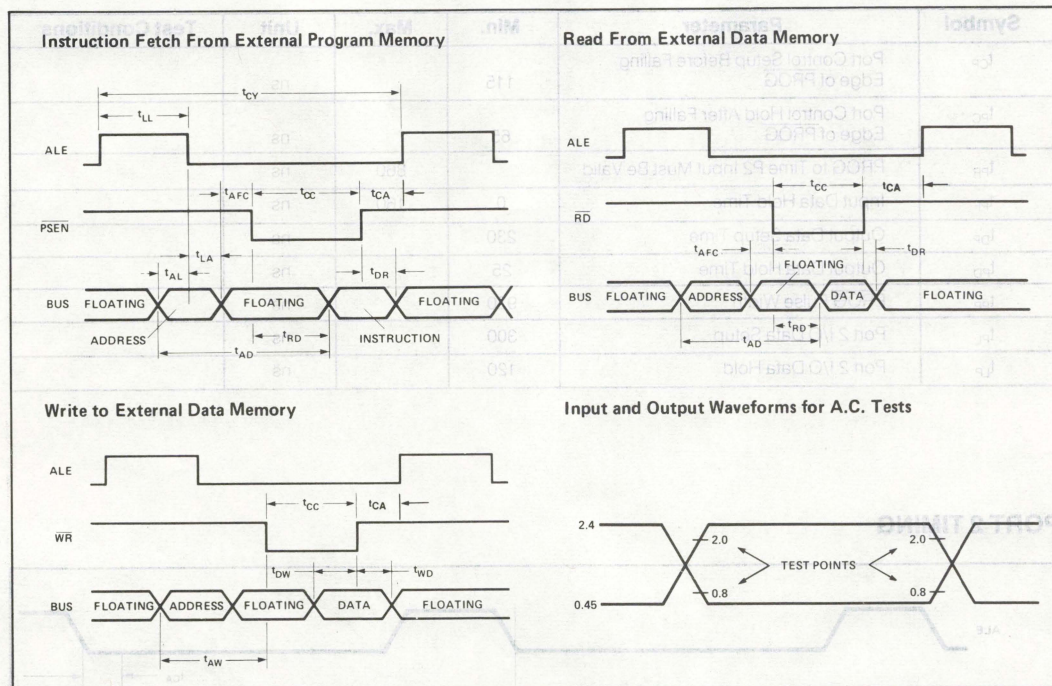
| Symbol   | Parameter                                      | Min. | Max. | Unit | Test Conditions |
|----------|--|------|------|------|-----------------|
| $t_{CP}$ | Port Control Setup Before Falling Edge of PROG | 115  |      | ns   |                 |
| $t_{PC}$ | Port Control Hold After Falling Edge of PROG   | 65   |      | ns   |                 |
| $t_{PR}$ | PROG to Time P2 Input Must Be Valid            |      | 860  | ns   |                 |
| $t_{PF}$ | Input Data Hold Time                           | 0    | 160  | ns   |                 |
| $t_{DP}$ | Output Data Setup Time                         | 230  |      | ns   |                 |
| $t_{PD}$ | Output Data Hold Time                          | 25   |      | ns   |                 |
| $t_{PP}$ | PROG Pulse Width                               | 920  |      | ns   |                 |
| $t_{PL}$ | Port 2 I/O Data Setup                          | 300  |      | ns   |                 |
| $t_{LP}$ | Port 2 I/O Data Hold                           | 120  |      | ns   |                 |

## PORT 2 TIMING





# WAVEFORMS



# A.C. CHARACTERISTICS $T_A = -55^{\circ}\text{C}$ to $125^{\circ}\text{C}$ , $V_{CC} = V_{DD} = +5\text{V} \pm 10\%$ , $V_{SS} = 0\text{V}$

| Symbol    | Parameter                          | M8048<br>M8035L |      | M8748 |      | Unit          | Conditions<br>(Note 1) |
|-----------|------------------------------------|-----------------|------|-------|------|---------------|------------------------|
|           |                                    | Min.            | Max. | Min.  | Max. |               |                        |
| $t_{LL}$  | ALE Pulse Width                    | 200             |      | 300   |      | ns            |                        |
| $t_{AL}$  | Address Setup to ALE               | 120             |      | 120   |      | ns            |                        |
| $t_{LA}$  | Address Hold from ALE              | 80              |      | 80    |      | ns            |                        |
| $t_{CC}$  | Control Pulse Width (PSEN, RD, WR) | 400             |      | 600   |      | ns            |                        |
| $t_{DW}$  | Data Setup before WR               | 420             |      | 600   |      | ns            |                        |
| $t_{WD}$  | Data Hold After WR                 | 80              |      | 120   |      | ns            | $C_L = 20\text{pF}$    |
| $t_{CY}$  | Cycle Time                         | 2.5             | 15.0 | 4.17  | 15.0 | $\mu\text{s}$ | (3.6 MHz XTAL 8748)    |
| $t_{DR}$  | Data Hold                          | 0               | 200  | 0     | 200  | ns            |                        |
| $t_{RD}$  | PSEN, RD to Data In                |                 | 400  |       | 600  | ns            |                        |
| $t_{AW}$  | Address Setup to WR                | 230             |      | 260   |      | ns            |                        |
| $t_{AD}$  | Address Setup to Data In           |                 | 600  |       | 900  | ns            |                        |
| $t_{AFC}$ | Address Float to RD, PSEN          | -40             |      | -60   |      | ns            |                        |
| $t_{CA}$  | Control Pulse to ALE               | 10              |      | 10    |      | ns            |                        |

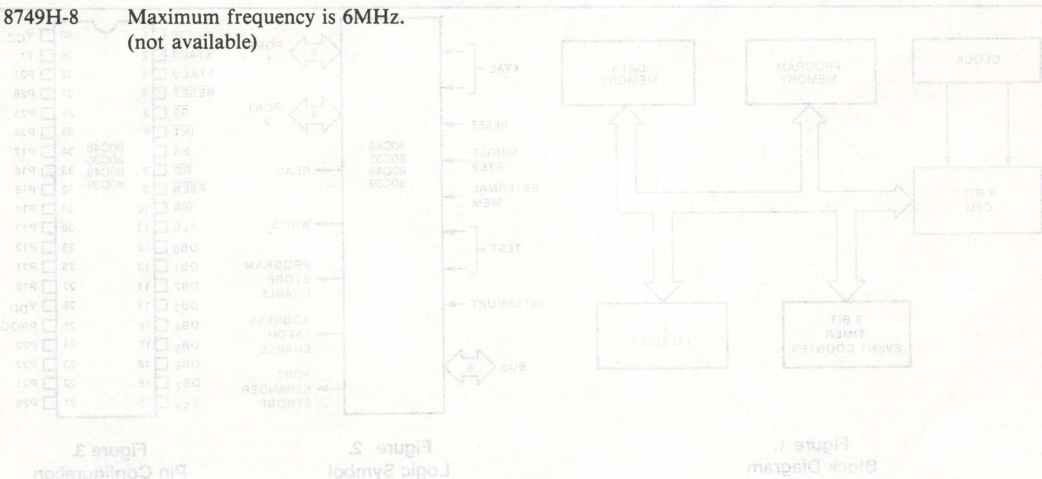
Note 1: Control outputs:  $C_L = 80\text{ pF}$   $t_{CY} = 2.5\text{ }\mu\text{s}$  for 8048/8035L  
 BUS Outputs:  $C_L = 150\text{ pF}$   $4.17\text{ }\mu\text{s}$  for 8748

## "DASH" NUMBERS AND THEIR MEANING

The purpose of this chart is to show the deviations from the Standard specifications of devices with a "dash" number. Unless otherwise mentioned, all the specifications of the standard part apply.

|          |   |
|----------|---|
| 8039-6   | Maximum frequency is 6MHz.<br>(not available)                           |
| 8035HL-1 | Maximum frequency is 11MHz.   |
| 8048H-1  | (No dash number means maximum frequency is 8MHz.)<br>(not available)    |
| 8748-4   | Does not work with 8243.<br>(not available)                             |
| 8748-6   | Temperature is 0°C to 55°C.<br>(not available)                          |
| 8035-8   | VCC = +5V ± 5%. Maximum   |
| 8748-8   | frequency is 3.6 MHz.   |
| 8749H-3  | VCC = +5V ± 5%.   |
| 8749H-5  | VCC = +5V ± 5%. Temperature range<br>is 0°C to 55°C.<br>(not available) |

8749H-8 Maximum frequency is 6MHz.  
(not available)





# 80C48/80C35/80C49/80C39 CHMOS SINGLE-COMPONENT 8-BIT MICROCOMPUTER

- 80C48/80C49 Low Power Mask Programmable ROM
- 80C35/80C39 Low Power, CPU only
- Pin-to-pin compatible with Intel's 8048/8035HL/8049H/8039HL
- 1.36  $\mu$ sec Instruction Cycle. All instructions 1 or 2 cycles
- Ability to maintain operation during AC power line interruptions
- Exit Idle mode with an external or internal interrupt signal
- Battery Operation
- 3 power consumption selections
  - Normal Operation: 15mA @ 11 MHz @ 6V
  - Idle Mode: 500  $\mu$ A @ 11 MHz @ 6V
  - Power down: 10  $\mu$ A @ 2.0V
- 11 MHz, TTL compatible operation;  $V_{CC} = 5V \pm 10\%$
- CMOS compatible operation;  $V_{CC} = 5V \pm 20\%$

Intel's 80C48/80C35/80C49/80C39 are low power, CHMOS versions of the popular MCS-48 HMOS family members. CHMOS is a technology built on HMOS II and features high resistivity P substrate, diffused N well, and scaled N and P channel devices. The 80C48/80C35/80C49/80C39 have been designed to provide low power consumption and high performance.

The 80C48/80C49 contains a 1K x 8/2K x 8 program memory, a 64 x 8/128 x 8 RAM data memory, 27 I/O lines, and an 8-bit timer/counter in addition to an on-board oscillator and clock circuits. For systems that require extra capability, the 80C48/80C49 can be expanded using CMOS external memories and MCS-80® and MCS-85® peripherals. The 80C35/80C39 is the equivalent of the 80C48/80C49 without program memory on-board.

The CHMOS design of the 80C48/80C49 opens new application areas that require battery operation, low power standby, wide voltage range, and the ability to maintain operation during AC power line interruptions. These applications include portable and hand-held instruments, telecommunications, consumer, and automotive.

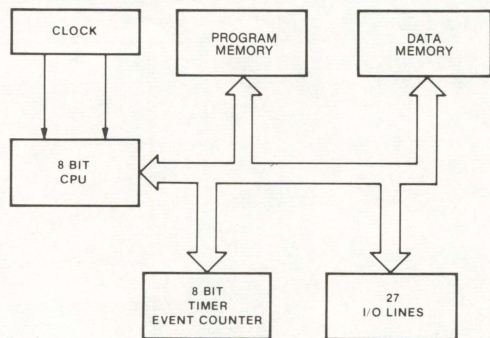


Figure 1.  
Block Diagram

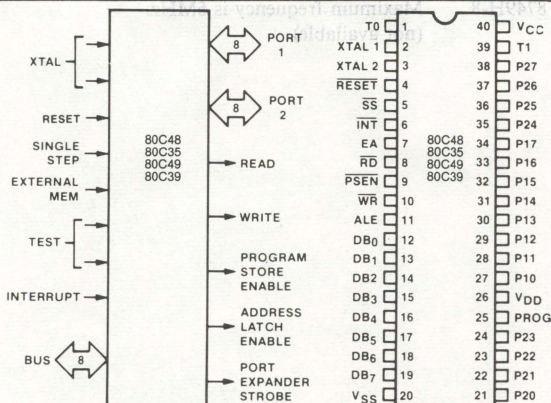


Figure 2.  
Logic Symbol

Figure 3.  
Pin Configuration



# ABSOLUTE MAXIMUM RATINGS\*

Ambient Temperature Under Bias ..... 0°C to 70°C  
 Storage Temperature ..... -65°C to + 150°C  
 Voltage On Any Pin With Respect  
     to Ground ..... -0.5V to V<sub>CC</sub>  
 Voltage on Any Pin with Respect  
     to V<sub>CC</sub> ..... V<sub>CC</sub> + 0.5V  
 Max Package Power Dissipation ..... 1.5 Watt

*\*NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

# D.C. CHARACTERISTICS (TA = 0°C to 70°C, V<sub>CC</sub> = V<sub>DD</sub> = 5V ± 10%, V<sub>SS</sub> = 0V)

| Symbol            | Parameter  | Limits |                  |                  | Unit           | Test Conditions   |
|-------------------|--|--------|------------------|------------------|----------------|---|
|                   |  | Min    | Typ              | Max              |                |   |
| V <sub>IL</sub>   | Input Low Voltage<br>(All Except RESET, X1, X2)            | -5     |                  | 8                | V              |   |
| V <sub>IL1</sub>  | Input Low Voltage<br>(RESET, X1, X2)                       | -5     |                  | 6                | V              |   |
| V <sub>IH</sub>   | Input High Voltage<br>(All Except XTAL1, XTAL2, RESET)     | 2.0    |                  | V <sub>CC</sub>  | V              |   |
| V <sub>IH1</sub>  | Input High Voltage (X1, X2, RESET)                         | 3.8    |                  | V <sub>CC</sub>  | V              |   |
| V <sub>OL</sub>   | Output Low Voltage (BUS)                                   |        |                  | .45              | V              | I <sub>OL</sub> = 2.0 mA                                  |
| V <sub>OL1</sub>  | Output Low Voltage<br>(RD, WR, PSEN, ALE)                  |        |                  | .45              | V              | I <sub>OL</sub> = 1.8 mA                                  |
| V <sub>OL2</sub>  | Output Low Voltage (PROG)                                  |        |                  | .45              | V              | I <sub>OL</sub> = 1.0 mA                                  |
| V <sub>OL3</sub>  | Output Low Voltage<br>(All Other Outputs)                  |        |                  | .45              | V              | I <sub>OL</sub> = 1.6 mA                                  |
| V <sub>OH</sub>   | Output High Voltage (BUS)                                  | 2.4    |                  |                  | V              | I <sub>OH</sub> = -400 μA                                 |
| V <sub>OH1</sub>  | Output High Voltage<br>(RD, WR, PSEN, ALE)                 | 2.4    |                  |                  | V              | I <sub>OH</sub> = -100 μA                                 |
| V <sub>OH2</sub>  | Output High Voltage<br>(All Other Outputs)                 | 2.4    |                  |                  | V              | I <sub>OH</sub> = -40 μA                                  |
| I <sub>L1</sub>   | Input Leakage Current (T1, INT)                            |        |                  | ± 10             | μA             | V <sub>SS</sub> ≤ V <sub>IN</sub> ≤ V <sub>CC</sub>       |
| I <sub>LI1</sub>  | Input Leakage Current<br>(P10-P17, P20-P27, EA, SS)        |        |                  | TBD              | μA             | V <sub>SS</sub> + .45 ≤ V <sub>IN</sub> ≤ V <sub>CC</sub> |
| I <sub>LI2</sub>  | Input leakage Current<br>(RESET)                           |        |                  | -300             | μA             | V <sub>SS</sub> ≤ V <sub>IN</sub> ≤ V <sub>IL1</sub>      |
| I <sub>L0</sub>   | Output Leakage Current (BUS, TO)<br>(High Impedance State) |        |                  | ± 10             | μA             | V <sub>SS</sub> + .45 ≤ V <sub>IN</sub> ≤ V <sub>CC</sub> |
| I <sub>DD</sub>   | V <sub>DD</sub> Supply Current                             |        |                  | 10               | μA             | V <sub>DD</sub> = 2.0V                                    |
| I <sub>CC</sub>   | Supply Current   |        |                  | 15<br>8.5<br>1.5 | mA<br>mA<br>mA | f = 11 MHz<br>f = 6 MHz<br>f = 1 MHz                      |
| I <sub>IDLE</sub> | Idle Mode Current  |        | 500<br>300<br>60 |                  | μA<br>μA<br>μA | f = 11MHz<br>f = 6 MHz<br>f = 1 MHz                       |



**Table 1. Added Instruction Set**

| Mnemonic | Description            | Byte | Cycles |
|----------|------------------------|------|--------|
| IDL      | Select Idle operation. | 1    | 1      |

## IDLE MODE DESCRIPTION

The 80C48/80C49, when placed into Idle mode, keeps the oscillator, the internal timer and the external interrupt and counter pins functioning and maintains the internal register and RAM status.

To place the 80C48/80C49 in Idle mode, a command instruction (op code 01H) is executed. To terminate Idle mode, interrupts must be enabled and an interrupt signal generated. There are two interrupt sources that can restore normal operation. One is an external signal applied to the interrupt pin. The other is from the overflow of the timer/counter. When either interrupt is invoked the CPU is taken out of Idle mode and vectors to the interrupt's service routine address. A reset signal will also take the processor out of Idle mode. Along with the Idle mode, the standard MCS-48 power-down mode is still maintained.

For Pin Description, see page 10-15.

For entire Instruction Set, see page 10-16.

For Timing Specifications, see page 10-18.

For Timing Waveforms, see page 10-19.

For recommended Oscillator configurations, see page 10-20.

| Symbol            | Parameter  | Min | Max |
|-------------------|--|-----|-----|
| V <sub>IL</sub>   | Input Low Voltage (All Except RESET, XT1, XT2)                       | 0.8 | 1.5 |
| V <sub>IL1</sub>  | Input Low Voltage (RESET, XT1, XT2)                                  | 0.8 | 1.5 |
| V <sub>IH</sub>   | Input High Voltage (All Except XTAL1, XTAL2, RESET)                  | 2.0 | 2.5 |
| V <sub>IH1</sub>  | Input High Voltage (XT1, XT2, RESET)                                 | 2.0 | 2.5 |
| V <sub>OL</sub>   | Output Low Voltage (BUS)   | 0.4 | 0.8 |
| V <sub>OL1</sub>  | Output Low Voltage (RD, WR, PSEN, ALE)                               | 0.4 | 0.8 |
| V <sub>OL2</sub>  | Output Low Voltage (PROG)  | 0.4 | 0.8 |
| V <sub>OL3</sub>  | Output Low Voltage (All Other Outputs)                               | 0.4 | 0.8 |
| V <sub>OH</sub>   | Output High Voltage (BUS)  | 2.4 | 2.5 |
| V <sub>OH1</sub>  | Output High Voltage (RD, WR, PSEN, ALE)                              | 2.4 | 2.5 |
| V <sub>OH2</sub>  | Output High Voltage (All Other Outputs)                              | 2.4 | 2.5 |
| I <sub>IL</sub>   | Input Leakage Current (T <sub>1</sub> )                              |     | 10  |
| I <sub>IL1</sub>  | Input Leakage Current (P10-P17, P20-P27, EA, SS)                     |     | 10  |
| I <sub>IL2</sub>  | Input Leakage Current (RESET)  |     | 10  |
| I <sub>LO</sub>   | Output Leakage Current (BUS, T <sub>0</sub> ) (High Impedance State) |     | 10  |
| I <sub>DD</sub>   | V <sub>DD</sub> Supply Current                                       |     | 10  |
| I <sub>CC</sub>   | Supply Current   |     | 10  |
| I <sub>IDLE</sub> | Idle Mode Current  |     | 10  |





## U.S. SALES OFFICES



3065 Bowers Avenue  
Santa Clara, California 95051  
Tel: (408) 987-8080  
TWX: 910-338-0026  
TELEX: 34-6372

### ALABAMA

Intel Corp.  
303 Williams Avenue, S.W.  
Suite 1422  
Huntsville 35801  
Tel: (205) 533-9353

### ARIZONA

Intel Corp.  
10210 N. 25th Avenue, Suite 11  
Phoenix 85021  
Tel: (602) 896-4980

### CALIFORNIA

Intel Corp.  
1010 Hurley Way  
Suite 300  
Sacramento 95825  
Tel: (916) 929-4078

Intel Corp.  
7670 Opportunity Rd.  
Suite 135  
San Diego 92111  
Tel: (714) 268-3563

Intel Corp.  
2000 East 4th Street  
Suite 100  
Santa Ana 92705  
Tel: (714) 835-9642  
TWX: 910-595-1114

Intel Corp.  
3375 Scott Blvd.  
Santa Clara 95051  
Tel: (408) 987-8086  
TWX: 910-339-9279  
910-338-0255  
Earle Associates, Inc.  
4617 Ruffner Street  
Suite 202  
San Diego 92111  
Tel: (714) 278-5441

Intel Corp.  
5530 Corbin Ave.  
Suite 120  
Tarzana 91356  
Tel: (213) 708-0333  
TWX: 910-495-2045

### COLORADO

Intel Corp.  
650 S. Cherry Street  
Suite 720  
Denver 80222  
Tel: (303) 321-8086  
TWX: 910-931-2289

### CONNECTICUT

Intel Corp.  
36 Padanaram Road  
Danbury 06810  
Tel: (203) 792-8366  
TWX: 910-456-1199

EMC Corp.  
48 Purnell Place  
Manchester 06040  
Tel: (203) 646-8085

EMC Corp.  
393 Center Street  
Wallingford, CT 06492  
Tel: (203) 265-9991

### FLORIDA

Intel Corp.  
1500 N.W. 62nd Street, Suite 104  
Ft. Lauderdale 33309  
Tel: (305) 771-0600  
TWX: 510-956-9407

Intel Corp.  
500 N. Maitland, Suite 205  
Maitland 32751  
Tel: (305) 628-9393  
TWX: 810-853-9219

### GEORGIA

Intel Corp.  
3300 Holcomb Bridge Rd.  
Norcross 30092  
Tel: (404) 449-0541

### ILLINOIS

Intel Corp.  
2550 Golf Road, Suite 815  
Rolling Meadows 60008  
Tel: (312) 981-7200  
TWX: 910-651-5881

### INDIANA

Intel Corp.  
9100 Purdue Rd.  
Suite 400  
Indianapolis 46268  
Tel: (317) 875-0623

### IOWA

Intel Corp.  
St. Andrews Building  
930 St. Andrews Drive N.E.  
Cedar Rapids 52402  
Tel: (319) 393-5510

### KANSAS

Intel Corp.  
9393 W. 110th St., Ste. 265  
Overland Park 66210  
Tel: (913) 642-8080

### MARYLAND

Intel Corp.  
7257 Parkway Drive  
Hanover 21076  
Tel: (301) 796-7500  
TWX: 710-862-1944

### MASSACHUSETTS

Intel Corp.  
27 Industrial Ave.  
Chelmsford 01824  
Tel: (617) 256-1800  
TWX: 710-343-6333

EMC Corp.  
381 Elliott Street  
Newton 02164  
Tel: (617) 244-4740  
TWX: 922-5331

### MICHIGAN

Intel Corp.  
26500 Northwestern Hwy.  
Suite 401  
Southfield 48075  
Tel: (313) 353-0920  
TWX: 810-244-4915

### MINNESOTA

Intel Corp.  
7401 Metro Blvd.  
Suite 355  
Edina 55435  
Tel: (612) 835-6722  
TWX: 910-578-2867

### MISSOURI

Intel Corp.  
502 Earth City Plaza  
Suite 121  
Earth City 63045  
Tel: (314) 291-1990

### NEW JERSEY

Intel Corp.  
Raritan Plaza  
2nd Floor  
Raritan Center  
Edison 08837  
Tel: (201) 225-3000  
TWX: 710-480-6238

### NEW MEXICO

BFA Corporation  
P.O. Box 1237  
Las Cruces 88001  
Tel: (505) 523-0601  
TWX: 910-983-0543

### BFA Corporation

3705 Westfield, N.E.  
Albuquerque 87111  
Tel: (505) 292-1212  
TWX: 910-989-1157

### NEW YORK

Intel Corp.  
300 Motor Pkwy.  
Hempstead 11787  
Tel: (516) 231-3300  
TWX: 510-227-6236

Intel Corp.  
80 Washington St.  
Poughkeepsie 12601  
Tel: (914) 473-2303

Intel Corp.  
211 White Spruce Blvd.  
Rochester 14623  
Tel: (716) 424-1050  
TWX: 510-253-7391

T-Squared  
4054 Newcourt Avenue  
Syracuse 13206  
Tel: (315) 463-8592  
TWX: 710-541-0554

T-Squared  
7353 Pittsford  
Victor Road  
Victor 14564  
Tel: (716) 924-9101  
TWX: 510-254-8542

### NORTH CAROLINA

Intel Corp.  
3306 W. Meadowview Rd.  
Suite 206  
Greensboro 27407  
Tel: (919) 294-1541

### OHIO

Intel Corp.  
6500 East Avenue  
Dayton 45414  
Tel: (513) 890-5350  
TWX: 810-450-2528

Intel Corp.  
Chagrin-Brainerd Bldg., No. 300  
28001 Chagrin Blvd.  
Cleveland 44122  
Tel: (216) 464-2736  
TWX: 810-427-9298

### OKLAHOMA

Intel Corp.  
4157 S. Harvard Ave.  
Suite 123  
Tulsa 74135  
Tel: (918) 744-8068

### OREGON

Intel Corp.  
10700 S.W. Beaverton  
Hillsdale Highway  
Suite 324  
Beaverton 97005  
Tel: (503) 641-8086  
TWX: 910-467-8781

### PENNSYLVANIA

Intel Corp.  
510 Pennsylvania Avenue  
Fort Washington 19034  
Tel: (215) 641-1000  
TWX: 510-661-2077

Intel Corp.  
201 Penn Center Boulevard  
Suite 301W  
Pittsburgh 15235  
Tel: (412) 823-4970

C.E.D. Electronics  
300 N. York Road  
Hartboro 19040  
Tel: (215) 674-9600

### TEXAS

Intel Corp.  
12300 Ford Rd.  
Suite 380  
Dallas 75234  
Tel: (214) 241-8087

Intel Corp.  
6420 Richmond Ave.  
Suite 280  
Houston 77057  
Tel: (713) 784-3400  
TWX: 910-881-2490

Industrial Digital Systems Corp.  
5925 Sovereign  
Suite 101  
Houston 77036  
Tel: (713) 988-9421

Intel Corp.  
313 E. Anderson Lane  
Suite 314  
Austin 78752  
Tel: (512) 454-3628

### UTAH

Intel Corp.  
268 West 400 South  
Salt Lake City 84101  
Tel: (801) 533-8086

### VIRGINIA

Intel Corp.  
1501 Santa Rosa Road  
Suite C-7  
Richmond 23288  
Tel: (804) 282-5668

### WASHINGTON

Intel Corp.  
Suite 114, Bldg. 3  
1603 116th Ave. N.E.  
Bellevue 98005  
Tel: (206) 453-8086  
TWX: 910-443-3002

### WISCONSIN

Intel Corp.  
150 S. Sunnyslope Rd.  
Brookfield 53005  
Tel: (414) 784-9060

\*Field Application Location





## U.S. DISTRIBUTORS

3065 Bowers Avenue  
Santa Clara, California 95051  
Tel: (408) 987-8080  
TWX: 910-338-0026  
TELEX: 34-6372

### ALABAMA

Arrow Electronics  
4171 University Dr.  
Suite 102 1/2 D  
Huntsville 35405  
Tel: (205) 830-1103  
†Hamilton/Avnet Electronics  
4812 Commercial Drive N.W.  
Huntsville 35805  
Tel: (205) 837-7210  
TWX: 810-726-2162  
†Pioneer/Huntsville  
1207 Putnam Drive N.W.  
Huntsville 35895  
Tel: (205) 837-9300  
TWX: 810-726-2197

### ARIZONA

†Hamilton/Avnet Electronics  
505 S. Madison Drive  
Tempe 85281  
Tel: (602) 231-5140  
TWX: 910-950-0077  
†Wyle Distribution Group  
8155 N. 24th Street  
Phoenix 85021  
Tel: (602) 249-2232  
TWX: 910-951-4282

### CALIFORNIA

Arrow Electronics, Inc.  
521 Weddell Dr.  
Sunnyvale 94085  
Tel: (408) 745-6600  
TWX: 910-339-9371  
†Avnet Electronics  
350 McCormick Avenue  
Costa Mesa 92626  
Tel: (714) 754-6051  
TWX: 910-595-1928  
Hamilton/Avnet Electronics  
19515 So. Vermont Ave.  
Torrance 90502  
Tel: (213) 558-2987  
Hamilton/Avnet Electronics  
1175 Bordeaux Dr.  
Sunnyvale 94086  
Tel: (408) 743-3300  
TWX: 910-339-9332  
†Hamilton/Avnet Electronics  
4545 Viewridge Ave  
San Diego 92123  
Tel: (714) 563-1969  
TWX: 910-335-1216  
†Hamilton/Avnet Electronics  
10912 W. Washington Blvd.  
Culver City 90230  
Tel: (213) 558-2193  
TWX: 910-340-6364 or 7073  
†Hamilton Electro Sales  
3170 Pullman Street  
Costa Mesa 92626  
Tel: (714) 641-4109  
TWX: 910-595-2638  
†Wyle Distribution Group  
124 Maryland Street  
El Segundo 90245  
Tel: (213) 322-8100  
TWX: 910-348-7140 or 7111  
†Wyle Distribution Group  
9525 Chesapeake Dr.  
San Diego 92123  
Tel: (714) 565-9171  
TWX: 910-335-1590  
†Wyle Distribution Group  
3000 Bowers Avenue  
Santa Clara 95052  
Tel: (408) 727-2500  
TWX: 910-338-0451 or 0296  
Wyle Distribution Group  
17872 Cowan Avenue  
Irvine 92713  
Tel: (714) 641-1800  
TWX: 910-595-1572

†Wyle Distribution Group  
451 E. 124th Avenue  
Thornton 80241  
Tel: (303) 457-9953  
TWX: 910-936-0770

†Hamilton/Avnet Electronics  
8765 E. Orchard Road  
Suite 708  
Englewood 80111  
Tel: (303) 740-1017  
TWX: 910-935-0787

†Wyle Distribution Group  
451 E. 124th Avenue  
Thornton 80241  
Tel: (303) 457-9953  
TWX: 910-936-0770

†Hamilton/Avnet Electronics  
8765 E. Orchard Road  
Suite 708  
Englewood 80111  
Tel: (303) 740-1017  
TWX: 910-935-0787

### COLORADO

†Wyle Distribution Group  
451 E. 124th Avenue  
Thornton 80241  
Tel: (303) 457-9953  
TWX: 910-936-0770  
†Hamilton/Avnet Electronics  
8765 E. Orchard Road  
Suite 708  
Englewood 80111  
Tel: (303) 740-1017  
TWX: 910-935-0787

### CONNECTICUT

†Arrow Electronics  
12 Beaumont Road  
Wallingford 06512  
Tel: (203) 265-7741  
TWX: 710-476-0162  
†Hamilton/Avnet Electronics  
Commerce Industrial Park  
Commerce Drive  
Danbury 06810  
Tel: (203) 797-2800  
TWX: 710-456-9974  
†Harvey Electronics  
112 Main Street  
Norwalk 06851  
Tel: (203) 853-1515  
TWX: 710-468-3373

### FLORIDA

†Arrow Electronics  
1001 N.W. 62nd Street  
Suite 108  
Ft. Lauderdale 33309  
Tel: (305) 776-7790  
TWX: 510-955-9456  
†Arrow Electronics  
115 Palm Bay Road, N.W.  
Suite 10, Bldg. 200  
Palm Bay 32909  
Tel: (305) 725-1480  
TWX: 510-959-6337  
†Hamilton/Avnet Electronics  
5800 Northwest 20th Ave.  
Ft. Lauderdale 33309  
Tel: (305) 971-2900  
TWX: 510-956-3097  
Hamilton/Avnet Electronics  
3197 Tech. Drive North  
St. Petersburg 33702  
Tel: (813) 576-3930  
TWX: 810-863-0374  
†Pioneer/Orlando  
6220 S. Orange Blossom Trail  
Suite 412  
Orlando 32809  
Tel: (305) 859-3600  
TWX: 810-850-0177

### GEORGIA

Arrow Electronics  
2979 Pacific Drive  
Norcross 30071  
Tel: (404) 449-8252  
TWX: 810-766-0439  
†Hamilton/Avnet Electronics  
5825 D. Peachtree Corners  
Norcross 30092  
Tel: (404) 447-7500  
TWX: 810-766-0432  
Pioneer/Georgia  
5835 B. Peachtree Corners E  
Norcross 30092  
Tel: (404) 448-1711  
TWX: 810-766-4515

### ILLINOIS

Arrow Electronics  
492 Lunt Avenue  
P.O. Box 94248  
Schaumburg 60193  
Tel: (312) 893-9420  
TWX: 910-291-3544  
†Hamilton/Avnet Electronics  
3901 N. 25th Avenue  
Schiller Park 60176  
Tel: (312) 678-5310  
TWX: 910-227-0060  
Pioneer/Chicago  
1551 Carmen Drive  
Elk Grove Village 60007  
Tel: (312) 437-9680  
TWX: 910-222-1834

### INDIANA

Arrow Electronics  
2718 Rand Road  
Indianapolis 46241  
(317) 243-9353  
TWX: 810-341-3119  
†Hamilton/Avnet Electronics  
485 Grady Drive  
Carmel 46032  
Tel: (317) 844-9333  
TWX: 810-260-3966

### INDIANA (Cont.)

Pioneer/Indiana  
5408 Castleplace Drive  
Indianapolis 46250  
Tel: (317) 849-7300  
TWX: 810-260-1794

### KANSAS

†Hamilton/Avnet Electronics  
9219 Quivera Road  
Overland Park 66215  
Tel: (913) 888-8500  
TWX: 910-743-0005  
†Component Specialties, Inc.  
8369 Nieman Road  
Lenexa 66214  
Tel: (913) 492-3555

### MARYLAND

†Hamilton/Avnet Electronics  
8622 Oak Hall Lane  
Columbia 21045  
Tel: (301) 995-3500  
TWX: 710-862-1861  
Mesa  
16021 Industrial Dr.  
Gaithersburg 20877  
Tel: (301) 948-4350  
†Pioneer  
9100 Gaither Road  
Gaithersburg 20780  
Tel: (301) 948-0710  
TWX: 710-828-0545

### MASSACHUSETTS

†Hamilton/Avnet Electronics  
50 Tower Office Park  
Woburn 01801  
Tel: (617) 935-9700  
TWX: 710-393-0382  
†Arrow Electronics  
Arrow Dr.  
Woburn 01801  
Tel: (617) 933-8130  
TWX: 710-393-6770  
Harvey/Boston  
44 Hartwell Ave.  
Lexington 02173  
Tel: (617) 863-1200  
TWX: 710-326-6617

### MICHIGAN

†Arrow Electronics  
3810 Varsity Drive  
Ann Arbor 48104  
Tel: (313) 971-8220  
TWX: 810-223-6020  
†Pioneer/Michigan  
13485 Stamford  
Livonia 48150  
Tel: (313) 525-1800  
TWX: 810-242-3271  
†Hamilton/Avnet Electronics  
32487 Schoolcraft Road  
Livonia 48150  
Tel: (313) 522-4700  
TWX: 810-242-8775

### MINNESOTA

†Arrow Electronics  
5230 W. 73rd Street  
Edina 55435  
Tel: (612) 830-1800  
TWX: 910-576-3125  
†Industrial Components  
5229 Edina Industrial Blvd.  
Minneapolis 55435  
Tel: (612) 831-2666  
TWX: 910-576-3153  
Hamilton/Avnet Electronics  
10300 Bren Rd. East  
Minnetonka 55343  
Tel: (612) 932-0668  
TWX: (612) 576-2720  
†Hamilton/Avnet Electronics  
7449 Cahill Road  
Edina 55435  
Tel: (612) 941-3801  
TWX: 910-576-2720

### MISSOURI

†Arrow Electronics  
2380 Schuetz  
St. Louis 63141  
Tel: (314) 567-6888  
†Hamilton/Avnet Electronics  
13743 Shoreline Ct.  
Earth City 63045  
Tel: (314) 344-1200  
TWX: 910-762-0684

### NEW HAMPSHIRE

†Arrow Electronics  
1 Perimeter Drive  
Manchester 03103  
Tel: (603) 668-9968  
TWX: 710-220-1684

### NEW JERSEY

†Arrow Electronics  
Pleasant Valley Avenue  
Moorestown 08057  
Tel: (215) 928-1800  
TWX: 710-897-0829  
†Arrow Electronics  
285 Midland Avenue  
Saddle Brook 07662  
Tel: (201) 797-5800  
TWX: 710-988-2206

### HAMILTON/AVNET ELECTRONICS

1 Keystone Ave.  
Bldg. 36  
Cherry Hill 08003  
Tel: (609) 424-0100  
TWX: 710-940-0262  
Hamilton/Avnet Electronics  
10 Industrial Road  
Fairfield 07006  
Tel: (201) 575-3390  
TWX: 710-734-4388  
†Harvey Electronics  
45 Route 46  
Pinebrook 07058  
Tel: (201) 227-1262  
TWX: 710-734-4382  
Measurement Technology Sales Corp.  
363 Route 46 W  
Fairfield 07006  
Tel: (201) 227-5552

### NEW MEXICO

†Alliance Electronics Inc.  
11030 Cochiti S.E.  
Albuquerque 87123  
Tel: (505) 292-3360  
TWX: 910-589-1151  
†Hamilton/Avnet Electronics  
2524 Baylor Drive S.E.  
Albuquerque 87119  
Tel: (505) 765-1500  
TWX: 910-989-0614

### NEW YORK

†Arrow Electronics  
900 Broad Hollow Rd.  
Farmingdale 11735  
Tel: (516) 694-6800  
TWX: 510-224-6494  
†Arrow Electronics  
3000 South Winton Road  
Rochester 14623  
Tel: (716) 275-0300  
TWX: 510-253-4766  
†Arrow Electronics  
7705 Maltage Drive  
Liverpool 13088  
Tel: (315) 652-1000  
TWX: 710-545-0230  
Arrow Electronics  
20 Oser Avenue  
Hauppauge 11787  
Tel: (516) 231-1000  
TWX: 510-227-6623  
†Hamilton/Avnet Electronics  
333 Metro Park  
Rochester 14623  
Tel: (716) 475-9130  
TWX: 510-253-5470  
†Hamilton/Avnet Electronics  
16 Corporate Circle  
E. Syracuse 13057  
Tel: (315) 437-2641  
TWX: 710-541-1560  
†Hamilton/Avnet Electronics  
5 Hub Drive  
Melville, Long Island 11746  
Tel: (516) 454-6000  
TWX: 510-224-6166  
Harvey Electronics  
P.O. Box 1208  
Binghamton 13902  
Tel: (607) 748-8211  
TWX: 510-252-0893





## U.S. DISTRIBUTORS

3065 Bowers Avenue  
Santa Clara, California 95051  
Tel: (408) 987-8080  
TWX: 910-338-0026  
TELEX: 34-6372

### NEW YORK (Cont.)

†Harvey Electronics  
60 Crossways Park West  
Woodbury, Long Island 11797  
Tel: (516) 921-8920  
TWX: 510-221-2184

Harvey/Rochester  
840 Fairport Park  
Fairport 14450  
Tel: (716) 381-7070  
TWX: 510-253-7001

Measurement Technology Sales Corp.

159 Northern Blvd.  
Greatneck 11021  
Tel: (516) 482-3500  
TWX: 510-223-0846

### NORTH CAROLINA

Arrow Electronics  
938 Burke Street  
Winston-Salem 27102  
Tel: (919) 725-8711  
TWX: 510-931-3169

†Hamilton/Avnet Electronics  
2803 Industrial Drive  
Raleigh 27609  
Tel: (919) 829-8030  
TWX: 510-928-1836

Pioneer/Carolina  
106 Industrial Ave.  
Greensboro 27406  
Tel: (919) 273-4441  
TWX: 510-925-1114

### OHIO

Arrow Electronics  
10 Knolkrick Dr.  
Reading 45237  
Tel: (513) 761-5432  
TWX: 810-461-2670

Arrow Electronics  
7620 McEwen Road  
Centerville 45459  
Tel: (513) 435-5551  
TWX: 810-459-1611

Arrow Electronics  
6238 Cochran Rd.  
Solon 44139  
Tel: (216) 248-3990  
TWX: 810-427-9409

†Hamilton/Avnet Electronics  
954 Senate Drive  
Dayton 45459  
Tel: (513) 433-0610  
TWX: 810-450-2531

†Hamilton/Avnet Electronics  
4588 Emery Industrial Parkway  
Warrensburg Heights 44128  
Tel: (216) 831-3500  
TWX: 810-427-9452

†Pioneer/Dayton  
4033 Interpoint Blvd.  
Dayton 45424  
Tel: (513) 236-9900  
TWX: 810-459-1622

†Pioneer/Cleveland  
4800 E. 131st Street  
Cleveland 44105  
Tel: (216) 587-3600  
TWX: 810-422-2211

### OKLAHOMA

†Components Specialties, Inc.  
7920 E. 40th Street  
Tulsa 74145  
Tel: (918) 664-2820  
TWX: 910-845-2215

### OREGON

†Almac/Strum Electronics  
8022 S.W. Nimbus, Bldg. 7  
Beaverton 97005  
Tel: (503) 641-9070  
TWX: 910-467-8743

†Hamilton/Avnet Electronics  
6024 S.W. Jean Rd.  
Bldg. C, Suite 10  
Lake Oswego 97034  
Tel: (503) 635-7848  
TWX: 910-455-8179

### PENNSYLVANIA

Arrow Electronics  
650 Seco Rd.  
Monroeville 15146  
Tel: (412) 856-7000

†Arrow Electronics  
650 Seco Rd.  
Monroeville 15146  
Tel: (412) 856-7000

Pioneer/Pittsburgh  
259 Kappa Drive  
Pittsburgh 15238  
Tel: (412) 782-2300  
TWX: 710-795-3122

Pioneer/Delaware Valley  
261 Gibraltar Road  
Horsesham 19044  
Tel: (215) 874-4000  
TWX: 510-665-6778

### TEXAS

Arrow Electronics  
13715 Gamma Road  
Dallas 75234  
Tel: (214) 386-7500  
TWX: 910-860-5377

Arrow Electronics, Inc.  
10700 Corporate Drive, Suite 100  
Stafford 77477  
Tel: (713) 491-4100  
TWX: 910-880-4439

†Component Specialties, Inc.  
8222 Jamestown Drive  
Suite 115  
Austin 78758  
Tel: (512) 837-8922  
TWX: 910-874-1320

†Component Specialties, Inc.  
10907 Shady Trail, Suite 101  
Dallas 75220  
Tel: (214) 357-6511  
TWX: 910-861-4999

†Component Specialties, Inc.  
8181 Commerce Park Drive, Suite 700  
Houston 77036  
Tel: (713) 771-7237  
TWX: 910-881-2422

Hamilton/Avnet Electronics  
10508A Boyer Blvd.  
Austin 78757  
Tel: (512) 837-8911  
TWX: 910-874-1319

†Hamilton/Avnet Electronics  
2111 W. Walnut Hill Lane  
Irving 75062  
Tel: (214) 659-4100  
TWX: 910-860-5929

†Hamilton/Avnet Electronics  
3939 Ann Arbor Drive  
Houston 77063  
Tel: (713) 780-1771  
TWX: 910-881-5523

Pioneer/Austin  
9901 Burnet Road  
Austin 78758  
Tel: (512) 835-4000  
TWX: 910-874-1323

### UTAH

†Hamilton/Avnet Electronics  
1585 West 2100 South  
Salt Lake City 84119  
Tel: (801) 972-2800  
TWX: 910-925-4018

### WASHINGTON

†Almac/Strum Electronics  
5811 Sixth Ave. South  
Seattle 98108  
Tel: (206) 753-2300  
TWX: 910-444-2067

†Hamilton/Avnet Electronics  
14212 N.E. 21st Street  
Bellevue 98005  
Tel: (206) 453-5844  
TWX: 910-443-2469

†Wyle Distribution Group  
1750 132nd Avenue N.E.  
Bellevue 98005  
Tel: (206) 453-8300  
TWX: 910-443-2526

### WISCONSIN

†Arrow Electronics  
430 W. Rausson Avenue  
Oak Creek 53154  
Tel: (414) 764-6600  
TWX: 910-262-1193

†Hamilton/Avnet Electronics  
2975 Moorland Road  
New Berlin 53151  
Tel: (414) 784-4510  
TWX: 910-262-1182

### VERMONT

†Hamilton/Avnet Electronics  
1585 West 2100 South  
Salt Lake City 84119  
Tel: (801) 972-2800  
TWX: 910-925-4018

### VERMONT

†Almac/Strum Electronics  
5811 Sixth Ave. South  
Seattle 98108  
Tel: (206) 753-2300  
TWX: 910-444-2067

### VERMONT

†Hamilton/Avnet Electronics  
14212 N.E. 21st Street  
Bellevue 98005  
Tel: (206) 453-5844  
TWX: 910-443-2469

### VERMONT

†Wyle Distribution Group  
1750 132nd Avenue N.E.  
Bellevue 98005  
Tel: (206) 453-8300  
TWX: 910-443-2526

### VERMONT

†Arrow Electronics  
430 W. Rausson Avenue  
Oak Creek 53154  
Tel: (414) 764-6600  
TWX: 910-262-1193

### VERMONT

†Hamilton/Avnet Electronics  
2975 Moorland Road  
New Berlin 53151  
Tel: (414) 784-4510  
TWX: 910-262-1182

### VERMONT

†Hamilton/Avnet Electronics  
1585 West 2100 South  
Salt Lake City 84119  
Tel: (801) 972-2800  
TWX: 910-925-4018

### VERMONT

†Almac/Strum Electronics  
5811 Sixth Ave. South  
Seattle 98108  
Tel: (206) 753-2300  
TWX: 910-444-2067

### VERMONT

†Hamilton/Avnet Electronics  
14212 N.E. 21st Street  
Bellevue 98005  
Tel: (206) 453-5844  
TWX: 910-443-2469

### VERMONT

†Wyle Distribution Group  
1750 132nd Avenue N.E.  
Bellevue 98005  
Tel: (206) 453-8300  
TWX: 910-443-2526

### VERMONT

†Arrow Electronics  
430 W. Rausson Avenue  
Oak Creek 53154  
Tel: (414) 764-6600  
TWX: 910-262-1193

### VERMONT

†Hamilton/Avnet Electronics  
2975 Moorland Road  
New Berlin 53151  
Tel: (414) 784-4510  
TWX: 910-262-1182

### VERMONT

†Hamilton/Avnet Electronics  
1585 West 2100 South  
Salt Lake City 84119  
Tel: (801) 972-2800  
TWX: 910-925-4018





## U.S. SERVICE OFFICES

3065 Bowers Avenue  
Santa Clara, California 95051  
Tel: (408) 987-8080  
TWX: 910-338-0026  
TELEX: 34-6372

### CALIFORNIA

Intel Corp.  
1601 Old Bayshore Hwy.  
Suite 345  
Burlingame 94010  
Tel: (415) 692-4762  
TWX: 910-375-3310

Intel Corp.  
2000 E. 4th Street  
Suite 110  
Santa Ana 92705  
Tel: (714) 835-2670  
TWX: 910-595-2475

Intel Corp.  
7670 Opportunity Road  
San Diego 92111  
Tel: (714) 268-3563

Intel Corp.  
5530 N. Corbin Ave.  
Suite 120  
Tarzana 91356  
Tel: (213) 708-0333

### COLORADO

Intel Corp.  
650 South Cherry  
Suite 720  
Denver 80222  
Tel: (303) 321-8086  
TWX: 910-931-2289

### CONNECTICUT

Intel Corp.  
36 Padanaram Rd.  
Danbury 06810  
Tel: (203) 792-8366

### FLORIDA

Intel Corp.  
1500 N.W. 62nd Street  
Suite 104  
Ft. Lauderdale 33309  
Tel: (305) 771-0600  
TWX: 510-956-9407

Intel Corp.  
500 N. Maitland Ave.  
Suite 205  
Maitland 32751  
Tel: (305) 628-2393  
TWX: 810-853-9219

Intel Corp.  
5151 Adanson St.  
Orlando 32804  
Tel: (305) 628-2393

### GEORGIA

Intel Corp.  
3300 Holcomb Bridge Rd. #225  
Norcross 30092  
Tel: (404) 449-0541

### ILLINOIS

Intel Corp.  
2550 Golf Road  
Suite 815  
Rolling Meadows 60008  
Tel: (312) 981-7230  
TWX: 910-253-1825

### KANSAS

Intel Corp.  
9393 W. 110th Street  
Suite 265  
Overland Park 66210  
Tel: (913) 642-8090

### MARYLAND

Intel Corp.  
7257 Parkway Drive  
Hancock 21076  
Tel: (301) 796-7500  
TWX: 710-862-1944

### MASSACHUSETTS

Intel Corp.  
27 Industrial Avenue  
Chelmsford 01824  
Tel: (617) 255-1800  
TWX: 710-343-6333

### MICHIGAN

Intel Corp.  
26500 Northwestern Hwy.  
Suite 401  
Southfield 48075  
Tel: (313) 353-0920  
TWX: 810-244-4915

### MINNESOTA

Intel Corp.  
7401 Metro Blvd.  
Suite 355  
Edina 55435  
Tel: (612) 835-6722  
TWX: 910-576-2867

### MISSOURI

Intel Corp.  
502 Earth City Plaza  
Suite 121  
Earth City 63045  
Tel: (314) 291-1990

### NEW JERSEY

Intel Corp.  
2460 Lemoine Avenue  
1st Floor  
Ft. Lee 07024  
Tel: (201) 947-6267  
TWX: 710-991-8593

### NEW YORK

Intel Corp.  
2255 Lyell Avenue  
Rochester 14606  
Tel: (716) 254-6120

### NORTH CAROLINA

Intel Corp.  
2306 W. Meadowview Rd.  
Suite 206  
Greensboro 27407  
Tel: (919) 294-1541

### OHIO

Intel Corp.  
Chagrin-Brainard Bldg. Suite 300  
28001 Chagrin Blvd.  
Cleveland 44122  
Tel: (216) 464-2736  
TWX: 810-427-9298

Intel Corp.  
6500 Poe Avenue  
Dayton 45414  
Tel: (513) 890-5350  
TWX: 810-450-2528

### OREGON

Intel Corp.  
10700 S.W. Beaverton-Hillsdale Hwy.  
Suite 22  
Beaverton 97005  
Tel: (503) 641-8086  
TWX: 910-467-8741

### PENNSYLVANIA

Intel Corp.  
500 Pennsylvania Avenue  
Fort Washington 19034  
Tel: (215) 641-1000  
TWX: 510-661-2077

Intel Corp.  
201 Penn Center Blvd.  
Suite 301 W.  
Pittsburgh 15235  
Tel: (412) 823-4970

### TEXAS

Intel Corp.  
313 E. Anderson Lane  
Suite 314  
Austin 78752  
Tel: (512) 454-8477  
TWX: 910-874-1347

Intel Corp.  
2925 L.B.J. Freeway  
Suite 175  
Dallas 75234  
Tel: (214) 241-2820  
TWX: 910-860-5617

Intel Corp.  
6420 Richmond Avenue  
Suite 280  
Houston 77057  
Tel: (713) 784-1300  
TWX: 910-881-2490

### VIRGINIA

Intel Corp.  
7700 Leesburg Pike  
Suite 412  
Falls Church 22043  
Tel: (703) 734-9707  
TWX: 710-931-0625

### WASHINGTON

Intel Corp.  
1603 118th Ave. N.E.  
Suite 114  
Bellevue 98005  
Tel: (206) 232-7823  
TWX: 910-443-3002

### WISCONSIN

Intel Corp.  
150 S. Sunnyslope Road  
Suite 148  
Brookfield 53005  
Tel: (414) 784-9060





# INTERNATIONAL SALES AND MARKETING OFFICES

3065 Bowers Avenue  
Santa Clara, California 95051  
Tel: (408) 987-8080  
TWX: 910-338-0026  
TELEX: 34-6372

## INTERNATIONAL DISTRIBUTORS/REPRESENTATIVES

### ARGENTINA

VLC S.R.L.  
Sarmiento 1630  
(1042) Buenos Aires  
Tel: 011 54 1 35 1201  
TELEX: Public Booth 9900 or 9901

### AUSTRALIA

A.J.F. Systems & Components Pty. Ltd.  
9 Barker Street  
Burwood  
Victoria 3125  
Tel: 61 3 288 9521  
TELEX: AA 31261

### AUSTRIA

Bacher Elektronische Geräte GmbH  
Rottenmühlgasse 26  
A 1120 Vienna  
Tel: (222) 83 56 46  
TELEX: 131532  
Rekrisk Elektronik Geräte GmbH  
Lichtensteinstrasse 9716  
A 1090 Vienna  
Tel: (222) 347846  
TELEX: 134759

### BELGIUM

Ineco Belgium S.A.  
Ave. des Croix de Guerre 94  
B 1120 Brussels  
Tel: (02) 216 01 60  
TELEX: 25441

### BRAZIL

Icotron S.A.  
05110 Av. Mutings 3650  
6 Andar  
Pirituba Sao Paulo  
Tel: 261-0211  
TELEX: 1122274/ICOTBR

### CHILE

DIN  
Av. Vic. MacKenna 204  
Casilla 6055  
Santiago  
Tel: 227 564  
TELEX: 352003

### CHINA

Schmidt & Co. Ltd.  
Wing On Centre, 28th Floor  
Connaught Road  
Hong Kong  
Tel: 011-852-5-455-644  
TELEX: 74766 SCHMC HK

### COLOMBIA

International Computer Machines  
Carrera 7 No. 72-34  
Aereo: Aereo 19403  
Bogota 1  
Tel: 211-7282  
TELEX: 45395 GAYCO

### CYPRUS

Cyprus Eltrom Electronics  
P.O. Box 5393  
Nicosia  
Tel: 21-27982

### DENMARK

ITT Multi Komponent  
Fabriksparken 31  
DK-2600 Glostrup  
Tel: (02) 45 66 45  
TX: 33355  
Scandinavian Semiconductor  
Supply A/S  
Nannasgade 18  
DK-2200 Copenhagen  
Tel: (01) 63 50 90  
TELEX: 19037

### FINLAND

Oy Fintronic AB  
Meikonkatu 24 A  
SF-00210  
Helsinki 21  
Tel: (0) 692 60 22  
TELEX: 124 224 Fron SF

### FRANCE

Celidis S.A.\*  
53, Rue Charles Frerot  
F-94250 Gentilly  
Tel: (01) 546 13 13  
TELEX: 200 485  
Jermyn S.A.  
rue Jules Ferry 35  
93170 Bagnolet  
Tel: (1) 859 04 04  
TELEX: 213810 F

Metrologie\*  
La Tour d'Asnières  
1 Avenue Laurent Cely  
92008-Asnières  
Tel: (1) 791 44 44  
TELEX: 611 448  
Tekelec Airtronic\*  
Cité des Bruyères  
Rue Carle Vernet  
F-92310 Sevres  
Tel: (01) 534 75 35  
TELEX: 204552

### GERMANY

Electronic 2000 Vertriebs GmbH  
Neumarkter Strasse 75  
D-8000 Munich 80  
Tel: (89) 43 40 61  
TELEX: 522561  
Jermyn GmbH  
Postfach 1180  
Schulstrasse 48  
D-6277 Camberg  
Tel: (6343) 4231  
TELEX: 484426  
Neye Enatechnik GmbH  
Schillerstrasse 14  
D-2085 Quickborn-Hamburg  
Tel: (4106) 6121  
TELEX: 213590

SPOERLE Systems GmbH  
Westfallendamm 84  
4600 Dortmund 1  
Tel: (0231) 57 93 15  
TELEX: 822555

### GREECE

American Technical Enterprises  
P.O. Box 156  
Athens  
Tel: 30-1-8811271  
TX: 30-1-8219470

### HONG KONG

Schmidt & Co. Ltd.  
Wing on Center, 28th Floor  
Connaught Road  
Hong Kong  
Tel: 5-455-644  
TELEX: 74766 SCHMC HK

### INDIA

Micronic Devices  
104/109C, Nirmal Industrial Estate  
Sion (E)  
Bombay 400022, India  
TW: 486-170  
TELEX: 011-5947 MDEV IN

### ISRAEL

Electronics Ltd.\*  
11 Rozans Street  
P.O. Box 93300  
Tel Aviv 61390  
Tel: (3) 47 51 51  
TELEX: 33638

### ITALY

Eledra S.S.P.A.\*  
Viale Elvezia, 18  
I 20154 Milano  
Tel: (2) 34 97 51  
TELEX: 332332

### JAPAN

Asahi Electronics Co. Ltd.  
KMM Bldg. Room 407  
2-14-1 Asano, Kokura  
Kita-Ku, Kitakyushu City 802  
Tel: (093) 511-5471  
TELEX: AECKY 7126-16  
Hamilton-Avnet Electronics Japan Ltd.  
YU and YOU Bldg. 1-4 Horidome-Cho  
Nishonashi Chuuo-Ku, Tokyo 103  
Tel: (03) 562-9911  
TELEX: 2523774

Ryoyo Electric Corp.  
Konwa Bldg.  
1-12-22, Tsukiji  
Chuo-Ku, Tokyo 104  
Tel: (03) 543-7711

Tokyo Electron Ltd.  
Shin Juku, Nomura Bldg.  
26-2 Nishi-Shin Juku-Ichome  
Shin Juku-Ku, Tokyo 160  
Tel: (03) 343-4411  
TELEX: 232-2220 LABTEL J

### KOREA

Koram Digital  
Room 909 Woonam Bldg.  
7, 1-KA Bongre-Dong  
Chung-Ku Seoul  
Tel: 238-123  
TELEX: K23542 HANSINT

### MEXICO

Provedora Electronica, S.A. (Proesa)  
Prol. Mactezuma Ole. 24  
Col. Romero de Terreros  
Apdo. Postal 21-139  
Mexico 21, D.F.  
Tel: 554-8300  
TELEX: 017-72402 PROVME

### NETHERLANDS

Ineco Nether. Comp. Sys. BV  
Turfsteekstraat 63  
P.O. Box 360  
NL Aalsmeer 1430  
Tel: (2977) 28855  
TELEX: 14693  
Koning & Hartman  
Koperwerf 30  
P.O. Box 4320  
2544 EN 's Gravenhage  
Tel: 31 (70) 210-101  
TELEX: 31528

### NEW ZEALAND

McLean Information Technology Ltd.  
P.O. Box 18-065  
Glenn Innes, Auckland, 6  
Tel: 587-037  
TELEX: NZ2763 KOSFY

### NORWAY

Nordisk Elektronic (Norge) A/S  
Postoffice Box 122  
Smedsvingen 4  
1364 Hvalstad  
Tel: (2) 786 210  
TELEX: 16963

### PORTUGAL

Ditram  
Componentes E Electronica LDA  
Av. Miguel Bombarda, 133  
P1000 Lisboa  
Tel: (1) 545 313  
TELEX: 14182 Brieks-P

### SINGAPORE

General Engineers Corporation Pte. Ltd.  
Bldg 3, Units 1003-1008, 10th Floor  
P.S.A. Multi-Storey Complex  
Pasir Panjang Road  
Singapore 0511  
Tel: 271-3163  
TELEX: RS23987 GENERCO

### SOUTH AFRICA

Electronic Building Elements, Pty. Ltd.  
P.O. Box 4609  
Hazelwood, Pretoria 0001  
Tel: 011-27-12-46-9221  
TELEX: 30181SA

### SPAIN

Interface S.A.  
Ronda San Pedro 22, 3°  
Barcelona 10  
Tel: (3) 301 78 51  
TWX: 51508  
ITT SESA  
Miguel Angel 23-3  
Madrid 10  
Tel: (1) 419 54 00  
TELEX: 27707

### SWEDEN

AB Gosta Backstrom  
Box 12009  
Alstranercatan 22  
S-10221 Stockholm 12  
Tel: (8) 541 080  
TELEX: 10135

Nordisk Electronic AB  
Box 27301  
S-10254 Stockholm  
Tel: (8) 635 040  
TELEX: 10547

### SWITZERLAND

Industra AG  
Gemsenstrasse 2  
Poststock 80 - 21190  
CH-8021 Zurich  
Tel: (1) 583 22 30  
TELEX: 56788

### TAIWAN

Taiwan Automation Co.\*  
3d Floor #75, Section 4  
Nanking East Road  
Taipei  
Tel: 771-0940  
TELEX: 11942 TAIAUTO

### TURKEY

Turkelek Electronics  
Aparuk Boulevard 169  
Ankara  
Tel: 189483

### UNITED KINGDOM

Bytech Ltd.  
Sutton Park Avenue  
Reading, Berkshire RG6 1A2  
Tel: (0734) 61 031  
TELEX: 848215

Comway Microsystems Ltd.

Market Street  
UK-Bracknell, Berkshire  
Tel: 44 (344) 51654  
TELEX: 847201

### DECADE Ltd.

Chiltern Lodge  
Haw Lane  
Bledlow Ridge  
Tel: (0240) 27219

Jermyn Industries (Mogul)  
Vestry Estate  
Sevenoaks, Kent  
Tel: (0732) 50144  
TELEX: 95142

### M.E.D.L.

East Lane Road  
North Wembley  
Middlesex HA9 7PP  
Tel: (01) 908 43 11  
TELEX: 28817

Rapid Recall Ltd.  
Rapid House/Denmark St  
High Wycombe  
Bucks, England HP11 2ER  
Tel: 44 494 26 271  
TELEX: 849439  
Rapid Recall  
28 High Street  
Nantwich CW5 5LJ  
Tel: 0270 627505  
TELEX: 36329

\*Field Application Location





# INTERNATIONAL SALES AND MARKETING OFFICES

3055 Bowers Avenue  
Santa Clara, California 95051  
Tel: (408) 987-8080  
TWX: 910-338-0026  
TELEX: 34-6372

## INTEL® MARKETING OFFICES

### AUSTRALIA

Intel Semiconductor Pty. Ltd.  
Suite 2, Level 15, North Point  
100 Miller Street  
North Sydney, NSW, 2060  
Tel: 450-847  
TELEX: AA 20097

### BELGIUM

Intel Corporation S.A.  
Rue du Moulin a Papier 51  
Boite 1  
B-1160 Brussels  
Tel: (02) 661 07 11  
TELEX: 24814

### DENMARK

Intel Denmark A/S  
Lyngbyvej 32F 2nd Floor  
DK-2100 Copenhagen East  
Tel: (01) 18 20 00  
TELEX: 19567

### FINLAND

Intel Finland OY  
Sentnerikuja 3  
SF-00400 Helsinki 40  
Tel: (0) 56244 55  
TELEX: 123 332

### FRANCE

Intel Corporation, S.A.R.L.  
5 Place de la Balance  
Silic 223  
94528 Rungis Cedex  
Tel: (01) 687 22 01  
TELEX: 270475

### GERMANY

Intel Semiconductor GmbH\*  
Seidstrasse 27  
D-8000 Muenchen 2  
Tel: (89) 53851  
TELEX: 523 177

Intel Semiconductor GmbH\*  
Mainzer Strasse 75  
D-6200 Wiesbaden 1  
Tel: (6121) 70 08 74  
TELEX: 04186183

Intel Semiconductor GmbH  
Verkaufsbüro Stuttgart  
Bruckstrasse 61  
7012 Feilbach  
West Germany  
Tel: (711) 58 00 82  
TELEX: 7254826

Intel Semiconductor GmbH\*  
Hohenzollern Strasse 5\*  
3000 Hannover 1  
Tel: (511) 32 70 81  
TELEX: 923625

Intel Semiconductor GmbH  
Vertriebsbüro Düsseldorf  
Ober-Ratherstrasse 2  
4000 Düsseldorf 30  
Tel: (211) 65 10 54  
TELEX: 8586977

### HONG KONG

Intel Semiconductor Ltd.  
99-105 Des Voeux Rd., Central  
18F, Unit B  
Hong Kong  
Tel: 5-450-847  
TELEX: 63869

### ISRAEL

Intel Semiconductor Ltd.\*  
P.O. Box 1659  
Haifa  
Tel: 4524 261  
TELEX: 46511

### ITALY

Intel Corporation Italia Spa\*  
Milanofon, Palazzo E  
20094 Assago (Milano)  
Tel: (02) 824 00 06  
TELEX: 315183 INTMIL

### JAPAN

Intel Japan K.K.\*  
5-6 Tokodai, Toyosato-machi  
Tsukuba-Gun, Ibaraki-Ken 300-26  
Tel: 029747-8511  
TELEX: 03656-161

### NETHERLANDS

Intel Semiconductor B.V.\*  
Cometongebouw  
Westblaak 106  
3012 Km Rotterdam  
Tel: (10) 149122  
TELEX: 22283

### NORWAY

Intel Norway A/S  
P.O. Box 92  
Hvamveien 4  
N-2013  
Skjetten  
Tel: (2) 742 420  
TELEX: 18018

### SWEDEN

Intel Sweden A.B.\*  
Box 20092  
Enghetsvagen 5  
S-16120 Bromma  
Tel: (08) 98 53 85  
TELEX: 12261

### SWITZERLAND

Intel Semiconductor A.G.\*  
Forchstrasse 95  
CH 8032 Zurich  
Tel: (01) 55 45 02  
TELEX: 557 89 ich ch

### UNITED KINGDOM

Intel Corporation (U.K.) Ltd.\*  
5 Hospital Street  
Nantwich, Cheshire CW5 5RE  
Tel: (0270) 626 560  
TELEX: 36620  
Intel Corporation (U.K.) Ltd.\*  
Pipers Way  
Swindon, Wiltshire SN3 1RJ  
Tel: (0793) 26 101  
TELEX: 444447 INT SWN

\*Field Application Location

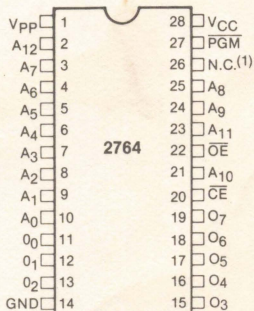
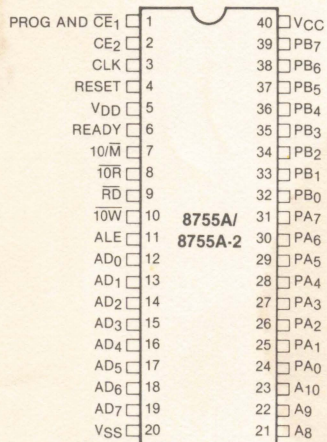
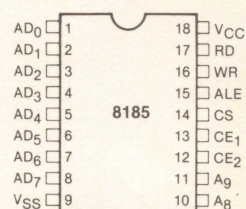
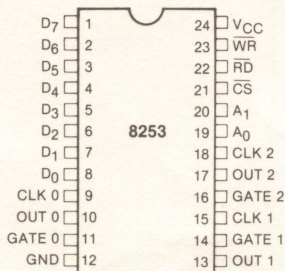
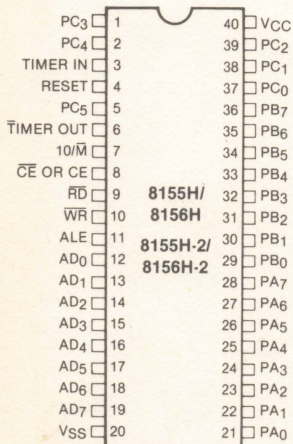
CD  
CI CO

1000/0000/0000  
7 F F

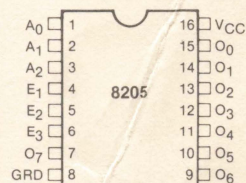
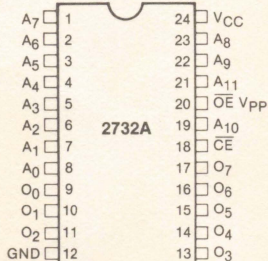
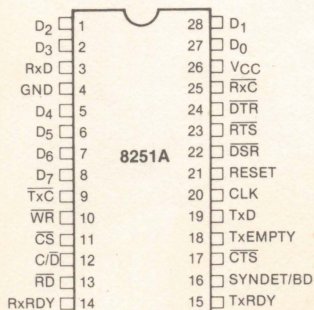
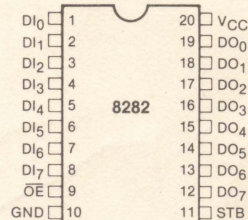
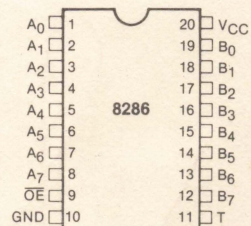
1

CD





(1) For total compatibility and upgradability from the 2732A and ROMs provide a trace to pin 26.







Intel Corporation  
3065 Bowers Avenue  
Santa Clara, CA 95051

Intel Corporation S.A.  
Parc Seny  
Rue du Moulin à Papier 51  
Boite 1  
B-1160 Brussels  
Belgium

Intel Japan K.K.  
5-6 Tokodai Toyosato-machi  
Tsukuba-gun, Ibaraki-ken 300-26  
Japan